PhD Dissertation

International Doctorate School in Information and Communication Technologies

DISI - University of Trento

Ultra-low-power Wireless Camera Network Nodes

Design and Performance Analysis

Leonardo Gasparini

Advisor:

Prof. Dario Petri

Università degli Studi di Trento

Co-Advisor:

Dr. Massimo Gottardi

Fondazione Bruno Kessler

April 2010

To the King, the Queen
and the Prince

Abstract

A methodology for designing Wireless Camera Network nodes featuring long lifetime is presented. Wireless Camera Networks may find widespread application in the fields of security, animal monitoring, elder care and many others. Unfortunately, their development is currently thwarted by the lack of nodes capable of operating autonomously for a long period of time when powered with a couple of AA batteries. In the proposed approach, the logic elements of a Wireless Camera Network node are clearly identified along with their requirements in terms of processing capabilities and power consumption. For each element, strategies leading to significant energy savings are proposed. In this context, the employment of a custom vision sensor and an efficient architecture are crucial. In order to validate the methodology, a prototype node is presented, mounting a smart sensor and a flash-based FPGA. The node implements a custom algorithm for counting people, a non trivial task requiring a considerable amount of on-board processing. The overall power consumption is limited to less than 5 mW, thus achieving a two orders of magnitude improvement with respect to the state of the art. By powering the system with two batteries providing 2200 mAh at 3.3 V, the expected lifetime of the system exceeds two months even in the worst-case scenario.

Keywords

Wireless Sensor Network, ultra-low-power, vision sensor, FPGA, power measurements, image processing

Contents

List of Tables

List of Figures

Chapter 1

Introduction

A Wireless Sensor Network (WSN) node (often referred to as just mote) is an autonomous computing system that is equipped with one or more sensors and that operates in absence of infrastructure, i.e., it is not connected to the power network and it communicates with other devices through a wireless channel [1]. Unlike the wired counterpart, with a WSN we can place the nodes very close to the phenomenon being monitored, without caring about the nodes positioning as long as the number of sensors provides enough information to filter out erroneous measurements. The acquired data are then processed on the node before being transmitted to a data collector which merges the pieces of information for analysis purposes or to generate an output, typically consisting on an alarm message or a command for some actuators.

Fields of application include, but are not limited to, the environmental monitoring (e.g., tracking the movement of animals, or acquiring information about the status of the soil for agricultural purposes), health control (e.g., recording the physiological activity of a living being, which moves and therefore cannot be connected to the infrastructure through wires), elder care (e.g., detecting people that fall on the ground and are not able to get up) and the military field (e.g., monitoring the status of troops and

acquiring information about unexplored terrains).

This thesis deals with Wireless Camera Networks (WCNs), a subclass of WSNs in which the sensing element is an imager. WCN may find widespread application in fields such as security, assisted living, road traffic monitoring, and natural sciences, and offer a number of advantages with respect to standard wired camera networks [2]. In particular, they are much easier to deploy and to remove, thus reducing installation time and costs. This is important for applications that require high density of placement, either to obtain many different views of the scene, or for increased robustness. Impromptu surveillance installations (for example, to monitor a building during a special event or animals in their natural habitat) also require fast installation, re-positioning and removal of a possibly large number of cameras, and thus would benefit from wireless technology. In-home elder care and home security may be facilitated by camera monitoring, and the use of wireless nodes would allow for discreet and unobtrusive installations at no costs, since the intervention of an operator is not needed [3]. WCNs represent a convenient solution also in those situations in which an external power source is unavailable or very expensive to provide, such as in borderlands and dangerous zones. For example, obtaining low voltage power supply from high-voltage power lines may increase the price of the wired surveillance system by a factor of 10. Finally, we note that WCNs are largely immune to failure of the power distribution system, and thus may support back-up of wired systems in the case of natural or man-made disasters.

Typically, a WSN node either operates on batteries or extracts the energy it needs from the environment, thus dictating the employment of ultra-low-power components to achieve a long lifetime. Of course, the limited energy resources affect the processing capabilities of the node, but this is not a problem whenever the amount of acquired data is not huge and

acquisitions are performed at low rates. This is the case for temperature, pressure and humidity measurements, for example. In general, the average power consumed by the sensing and processing units is limited to less than a milliwatt and most of the power is spent for wireless communication.

This condition is not true anymore when the sensing unit is a video sensor, which is much more complex and power-demanding than other sensors such as the ones that measure temperature, pressure or humidity. Not only the sensing process itself is more expensive (as the imager is composed of an array of thousands or millions of sensors), but also data buffering, processing and transmission require much more power. At the moment, no feasible energy scavenging system exists that can provide enough power and the use of batteries seem the only viable solution. Unfortunately, in this context, batteries have a very limited energy budget, resulting in a trade-off between the system's lifetime (typically dictated by the application) and the desired computational power to process the data. Thus, video nodes must be designed by carefully selecting the hardware components and by producing high-efficiency embedded software.

In the past decade, several WCN nodes have been proposed [4, 5, 6, 7, 8], but they have either high power consumption or limited performance. These systems employ standard imaging sensors, which are typically designed "for humans", providing high resolution images with several bits per pixel. Better energy efficiency is achieved by decreasing the sensor resolution, as long as the resolution is enough for the task at hand [5, 7]. Nevertheless, transmission of the full video stream via IEEE 802.11 or Bluetooth still requires too much power to be viable. Hence, a certain amount of on-board processing is necessary, but also data processing is expensive in terms of power, and it introduces a latency that may reduce the effective frame rate.

The source to the problem of the above mentioned systems lies in the

use of standard imagers, which are not necessarily the optimal solution for a WCN node. In fact, in most cases only particular portions of the image (such as moving areas, or objects in the foreground) or features (edges, corners, histograms of brightness or color) are of interest. This suggests the use of custom imagers that only acquire the data one is interested in, thus saving energy in the acquisition and in the processing phases. In such a framework, a new approach to the design of video sensors has been recently proposed by Teixeira *et al.* [9], one that attempts to limit power requirements by implementing the processing (or some parts of it) directly on-chip, thus allowing the entire system to save considerable amount of power and time. One step further is presented in [10], where the authors suggest the development of sensors able to perform object detection autonomously, with a very limited external control.

This thesis proposes a design methodology for very low-power WCN nodes based on this kind of sensors [11]. By pre-processing the images directly on chip, less data is generated, thus reducing power at multiple levels (less data to transfer, buffer, and process). Nevertheless, the problem is still challenging because even running algorithms on already pre-processed data is a power consuming task. The components present on the node need to operate with a very limited energy budget and at the same time to have enough computational capabilities to execute the algorithms in a short period of time. The solution to the problem is not unique and needs to be faced from different standpoints. On the one hand, one must identify which are the components that best fit the case at hand, having clear in mind how to exploit the resources they provide; on the other hand, one must implement efficient algorithms on the basis of the images generated by the sensor and of the available hardware, power and timing resources. These are concurrent aspects that one has to consider as a whole. We have analyzed the main logic components of a WCN node pointing out

the characteristics they need to have in order to maximize the network lifetime, as long as the provided performance suits the application. Accordingly, we have proposed power-saving strategies that guide the design process by defining the choices about the hardware components and the firmware implementation. The key element lays in developing within the firmware techniques to exploit the low power capabilities offered by the hardware.

To validate the proposed methodology, we have created a prototype WCN node. Such prototype exploits an ultra-low-power binary contrast-based imaging sensor that also features on-chip frame differencing [10]. The node core is represented by a flash-based Field Programmable Gate Array, which manages the system and processes the data generated by the sensor. Techniques to limit power consumption have been applied in a real situation. We have designed the node to operate as a People Counter, which has been used as a case study. In the end, the prototype's power consumption has been analyzed, and we have proved that the node can operate autonomously for more than a month when powered with a couple of standard batteries, which is two order of magnitudes longer than the other camera nodes proposed by the research community.

The proposed approach is suitable for many applications, but we are aware of its limitations. Of course, we cannot expect an ultra-low-power node to provide as many details as a standard wired video network. Nevertheless, our solution may play a fundamental role also in an advanced wireless surveillance system, in which the network is constituted by heterogeneous nodes [12]. Such kind of networks are organized hierarchically in multiple tiers, in which nodes belonging to the lower tiers are in charge of detecting the presence of some event of interest in the scene and possibly of localizing it. Such nodes are always active and therefore must consume very little power. When something relevant takes place, a wake-up trigger

is generated for the higher tiers, where nodes equipped with high resolution imagers process the frames with complex algorithms and, in case the situation requires the intervention of a human operator, transmit them. In this way, power is optimized since intensive processing is performed only when really needed. In this context, the lowest tiers determine the lifetime of the network, since they operate unceasingly until their power source is depleted, but, at the moment, no camera node seems to have the characteristics to fill such a role in the network. As a result, WCNs' lifetime is limited to few days, thus obstructing the development of this technology, especially in the commercial sector. We believe that our solution represents the answer to these needs.

The thesis is organized as follow. In Ch. 2 we provide the state of the art about low power WSN systems which are commercially available and about other hardware solutions which one needs to take into account when designing an ultra-low-power WCN node. We also provide some background about other existing WCN nodes as reported in the literature, pointing out their strong and weak points. In Ch. 3, we describe our general design methodology for an ultra-low-power WCN node, both from the hardware and the firmware points of view. We provide the detailed power model for a generic node designed according to our method as well. In Ch. 4 we describe the architecture of the prototype node that we developed. The chapter is structured following the design method, getting into the details of the hardware and firmware aspects. Then, the implemented People Counter is described firstly from a high level point of view, and then in its hardware implementation. Power requirement analysis and lifetime estimate of this node are presented in the last part of the chapter. Ch. 5 has the conclusions.

Chapter 2

State of the Art and Related Work

This thesis main goal is to provide a method for designing WCN nodes capable of operating for long periods with a very limited power source. In pursuing this goal , in this chapter we will concentrate on the system's hardware configuration rather than on other aspects such as the network communication architecture, sensor network topology, energy scavenging methods, etc. Generally speaking, "a sensor node is made up of four basic components: a sensing unit, a processing unit, a transceiver unit and a power unit" [1]. In the following we will consider just the first three of them, since those are the ones of major interest from the system designer's point of view. This is because we expect the system to guarantee proper functionality continuously for a certain period of time, disregarding what the power source is, as long as it provides a given (small) capacity.

Several solutions are offered by the market and by the research community to develop a wireless node. One may choose to buy a commercial, off-the-shelf (COTS) node platform (typically including the processing unit, the transceiver and expansion ports for the sensors), which provides an integrated environment for fast and easy development. Nevertheless WCNs, and ultra-low-power WCNs in particular, have critical requirements with respect to the technology currently available, and COTS platforms either

do not provide sufficient processing power to run the algorithms or the available processor are too expensive in terms of energy to run continuously, clocked at high frequency. Therefore researchers opted for developing their own custom platform, thus achieving a node highly fitted to the application. Such platforms may be either stand-alone modules including all the four basic components, or daughter boards which integrate the sensor and processing units and have to be attached to a COTS host-node, which takes care of interfacing the sensor with the network. Nevertheless, we will see that there is a lack of ultra-low-power nodes in the field, and no WCN node, an therefore no WCN either, seems to be able to operate on batteries for a long time without replacing the power source.

In this chapter we are going to present a roundup about the state of the art in individual devices and in host-nodes for WCNs, and discuss about the related work in the field. This chapter is organized as follows. In the first part we will briefly introduce the most popular low-power cameras, field programmable devices and wireless transceivers, and general purpose WSN host-nodes available in the market. Then we will concentrate on complete WCN nodes present in the literature that brought original ideas from the architectural point of view, aiming at lowering power consumption and improving power efficiency, without compromising the network effectiveness. Since a great variety of hardware platforms were developed by the research community in the past 10 years, we will concentrate on the few nodes which in our opinion effectively introduced important improvements in the field. The other relevant nodes will be briefly mentioned at the end of the chapter.

2.1 Commercial, Off-the-Shelf Products

2.1.1 Image sensors

As we will see later in this chapter, most of WCN nodes employ OmniVision CMOS camera modules. OmniVision is a company founded in 1995 and headquartered in Santa Clara (CA, USA), that designs CMOS sensors and camera modules. Products range from sub-VGA cameras to HD devices. Within the low-range product series, OmniVision cameras provide low power consumption, around 100 mW at quite high frame rates, 30 to 60 fps. Acquisitions at different resolutions are possible too. Moreover, their camera modules provide easy interfacing, automatic image control functions and basic image manipulation inside the module itself. As an example, the OV6680 [13] includes a CMOS camera providing images at a resolution that ranges from CIF up to 400×400 pixels, and exhibiting an active power consumption of 70 mW at 30 fps, CIF format, and a standby current lower than 20 μA.

Other solutions that suit WCN applications are all those cameras that can be mounted on mid-range cellphones, such as the old VS6524 [14], released by ST MicroElectronics in 2004.

2.1.2 Processing devices

The market of processing devices is huge. A great multitude of solutions from several companies are available to design a camera node. Mainly, we can distinguish between four classes of processing devices: low-power microcontrollers, high performance ones/embedded processors, Complex Programmable Logic Devices and low-power Field Programmable Gate Arrays.

Both microcontrollers and embedded processors are based on a single

microprocessor executing a software stored into a ROM. The more advanced modules often run an Operating System (OS), specifically designed for embedded systems, that manages the processes on that processor, including the access to hardware resources. Lately, multi-core processors have been introduced in the market, slightly relieving the "one instruction at a time" limitation introduced by the presence of a single processing core.

A MicroController Unit (MCU) is a single integrated circuit that contains a processor, memory and a set of peripherals such as counters, watchdog timers[1], multipliers, dividers, ADCs and DACs, and many more. Low power modes are typically supported, too. The number of available configurations are countless. One may choose a MCU based on a 8-, 16-, or 32-bit architecture and clocked at a speed that ranges from a few kilohertz's to hundreds of megahertz's; MCUs are available with different amounts of flash ROM to store the program instructions and SRAM for data buffering, ranging from a few bytes to a few megabytes. Power consumption varies accordingly, from less than a milliwatt, as in the 8 bit PIC10 series by Microchip Technologies [15], to hundreds of milliwatts, as in the dual core 32 bit Qorivva device by Freescale Semiconductor [16].

Embedded processors can be seen as high performance MCUs or as low-power processors, and in fact they represent the boundary between the two classes. An example is given by Marvell's PXA3xx family of processors [17], which can operate at more than 800 MHz clock frequency, include up to 768 Kbytes of internal SRAM and $32+32$ KB of cache for instructions and data. Available peripherals include counters, watchdog timers and pulse width modulators and the chip features all types of interfaces, including one for a camera. Of course all these properties come at the price of a higher power consumption with respect to low-power MCUs: at maximum

[1] A watchdog timer is a timer that generates a system reset every time it overflows. It is used to avoid the MCU to be stuck into an infinite loop due to some unexpected behavior. Its the programmer's task to reset the watchdog timer periodically to avoid a reset.

speed the processor consumes several hundreds of milliwatts. Even more powerful devices are considered embedded processors; among these there is the Intel Atom Processor [18], but in this case consumption increases significantly, exceeding the watt.

Complex Programmable Logic Devices (CPLDs) and Field Programmable Gate Arrays (FPGAs) are two classes of Programmable Logic Devices (PLDs) which are typically programmed in Hardware Description Language (HDL), namely VHDL or Verilog. They contain a matrix of configurable logic blocks, which in an FPGA typically contain Look-Up Tables (LUTs), flip-flops, MUltipleXers (MUXs) and other logic. Such basic elements can be configured and routed "on the field" to carry out the required task. Since VHDL and Verilog are languages describing hardware at a lower abstraction lever with respect to programming languages such as Java, C and even Assembly, writing the code for a CPLD/FPGA is much more time consuming than a processor. Nevertheless such devices allow to overcome many of the limitations that processors have, even the multi-core ones. Consider a simple processor: it has a well defined architecture that we cannot change, and the processor core executes all the instructions sequentially, one after the other. Moreover, only one peripheral at a time can write on the bus, since it is shared. As a consequence, a processor core executes far less than one instruction per clock cycle. Programmable logic devices are different in the sense that they allow to configure the internal logic, and it is the programmer's task to create the architecture of the processing unit. This allow to write highly optimized code, and to design processing units which overcome the "one clock-cycle per instruction" limit imposed by processors, thus boosting the system's performance. Furthermore, the manufacturers of PLDs provide tools for the generation of complex and customizable structures such as First In, First Out (FIFO) memories, filters, and even processors, in a completely

automatic fashion. Since these structures are built exploiting the configurable blocks, they are called "soft"-cores, in contrast with the "hard-", unchangeable counterparts, printed on silicon.

CPLDs and FPGAs are built on different technologies and their basic blocks are designed with different goals in mind; from a general, high level perspective, FPGAs, even the low-power ones, allow for much bigger designs and improved performance with respect to CPLDs, which, on the other hand, are cheaper and consume less power. Moreover, FPGAs are typically volatile, therefore they require some additional logic when they are turned on in order to be programmed, while CPLDs are non-volatile, thus enabling a significant reduction in system complexity. Lately, flash-based FPGAs has been introduced in the market, filling the gap between standard FPGAs and CPLDs.

From the perspective of WCNs, each class has its own pros and cons, which we try to summarize here and in Tab. 2.1.

- Processor based devices are relatively easy to program, especially when there is an OS that manages the resources to avoid conflicts; on the other side, the presence of a few processing elements and a single bus introduces significantly latency delays, which could severely limit the performance of a WCN node. Another negative aspect is represented by the limited amount of memory which typically characterizes these devices, thus forcing the employment of an additional, external element serving both as interface between the sensor and the processor and as a temporary buffer while processing the data.

- PLDs allow to generate an architecture fitted to the application and in which tasks are executed in parallel without the risk of long latency delays and conflicts among the implemented entities. As a drawback, programming such devices is very difficult, and the time required for

Table 2.1: Processing devices - Summary table

	Processor based	*PLDs*
Low power consumption	MCUs	CPLDs
High performance	Embedded processors	Low-power FPGAs

writing the code increases exponentially with the complexity of the design. Nevertheless, manufacturers provide soft-cores which are becoming easier and easier to use, thus limiting the difficulties introduced by the use of HDL.

- Low-power MCUs and CPLDs consume very little power, and have been widely used in standard WSNs. Nevertheless, their limited processing capabilities may constitute a problem in WCNs, due to the huge amount of data (represented by the images) that needs to be processed.

- Embedded processors and low-power FPGAs seem to respect the high processing power requirements set by WCNs, but may not be able to operate for a long period of time due to their elevate power consumption.

- Mixed solutions, such as the combination of a low-power MCU and an FPGA, introduce more flexibility and allow to overcome the limitations introduced by the single devices; the disadvantage lays in the fact that the entire system is more complex to design and power consumption may increase due to the higher activity on input and output pads.

In conclusion, the choice of the hardware devices responsible for processing strongly depends on the amount of data generated by the sensor,

the complexity of the algorithms that need to be executed by the node, and by requirements in terms of acquisition rate, latency and lifetime imposed by the application.

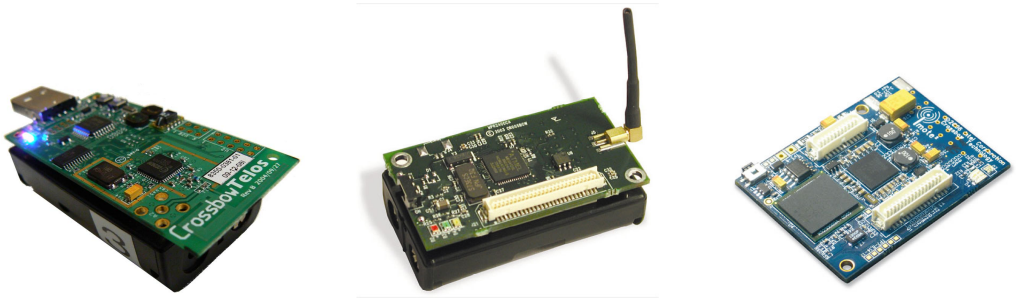
2.1.3 Wireless transceivers

There are mainly four wireless technologies that are considered in the development of low-power systems: ZigBee, Bluetooth, Wi-Fi, and Ultra Wide Band [19]. The ZigBee protocol is universally accepted as the protocol that best suits for WSNs. This is because it exhibits ultra-low-power capabilities and at the same time enables to build wireless mesh networks, i.e., networks in which all devices are actively involved in the network construction, as "they dynamically join the network, acting as both user terminals and routers for other devices, consequently further extending network coverage" [20]. The Bluetooth protocol provides higher data rates ($1-3$ Mbps vs 250 kbps), but is more oriented towards "short-range and cheap devices to replace cables for computer peripherals" [19]. This is because Bluetooth does not support Mesh networks as ZibBee, but piconets and scatternets. Piconets are networks in which one master device manages the connection among itself and several slave devices. In this configuration, each slave device can communicate only with the master. Scatternets are composed by several interconnected piconets. The rigidity imposed by the master-slave configuration makes Bluetooth unsuited for WSNs. Wi-Fi and UWB may be of interest for camera-based SNs. In fact, ZigBee is efficient only when the amount of transmitted/received data is low, so that the transceiver is mostly idle. But when we need to transmit images or, even more, to stream videos, Wi-Fi and UWB are much more power efficient [21, 22], i.e., they exhibit a lower consumption per bit transmitted. Nevertheless, these products exhibit a higher absolute power consumption, which makes them unsuitable for ultra-low-power WCN nodes.

The market leader of low-power wireless transceivers, based on the ZigBee/IEEE 802.15.4 standards, is ChipCon. ChipCon is a Norvegian Company founded in 1996, and acquired by Texas Instruments in 2006, which became popular for its CC1xxx and CC2xxx families of low-power wireless transceivers. The CC1xxx family operates in the $300-1000$ MHz frequency range while the CC2xxx one is IEEE 802.15.4 compliant [23], operating at 2.4 GHz. The most popular chips are the CC1000 [24] and the CC2420 [25], which are widely employed in WSN nodes thanks to their low current consumption (7.4/10.4/0.8 mA for the former and 19.7/17.4/0.8 mA for the latter, while receiving, transmitting and being idle respectively), support for low-power modes, high flexibility in the configuration and small form factor. The provided data rates are 76.8 kbps for the CC1000 and 256 kbps for the CC2420. Lately, Texas Instruments released the CC2500 [26], an improved version of the CC2420, achieving higher power efficiency.

Another important player is Nordic Semiconductor. Nordic Semiconductor ASA is a Norwegian company headquartered in Oslo, that operates in the field of wireless communication and multimedia. This company proposes ultra-low-power solutions that exhibit performance similar to ChipCon devices; nevertheless the are not part of the ZigBee Alliance, meaning that they developed their own technology, still based on the IEEE 802.15.4 standard.

Texas Instruments manufactures Bluetooth devices also, such as the CC2540 [27]. Power consumption is in the same order of magnitude as the ZigBee-based solutions; in fact it draws 19.4 mA in reception mode, 24 mA in transmission mode and much less than 1 mA when exploiting one of the low-power modes. Recently, a new, Bluetooth-based, ultra-low-power device has been introduced by Nordic Semiconductor: the nRF8001 [28]. This device consumes very little power, namely 14.5/13/1.6 mA when receiving, transmitting and while in stand-by respectively.



(a) Tmote Sky/TelosB (b) MicaZ (c) iMote2

Figure 2.1: Three popular COTS motes by Moteiv/Memsic

2.1.4 General purpose motes

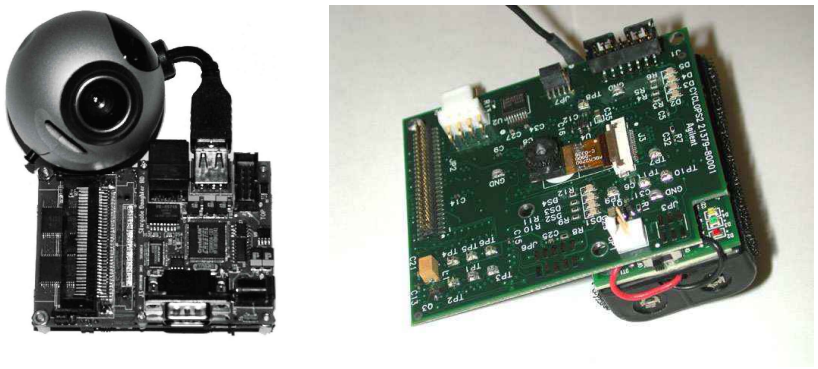
Moteiv, from San Francisco (CA, USA), is a company that operates in the field of WSN. Moteiv main product is the Tmote Sky. Moteiv was founded by three University of California, Berkeley (UCB) Ph.D. students in 2003. Since 2006 Moteiv became a subsidiary of Sentilla, a Redwood City (CA, USA)-based company operating in the field of energy. Memsic is another company from Andover (MA, USA) founded in 1999, which is currently the world leading supplier of wireless sensor technology. They provide several wireless modules as host nodes for WSN: TelosB, MICA2 and MICAz, and iMote2, which were originally manufactured by CrossBow, another American company operating in the field of smart-sensor technology.

Tmote Sky [29] is an ultra-low power host-mote, equipped with a Texas Instruments MSP430 Microcontroller [30] and a Chipcon CC2420 RF transceiver. The MSP430 is based on a 16 bit RISC architecture and is designed to operate at low frequencies, in the order of 1 MHz. It is widely used in WSNs, since it provides very high flexibility in terms of power consumption: five different power modes are supported, with a supply current ranging from 330 μA when the processing core is running at 1 MHz, down to 0.5 μA when most of the system is disabled. In addition, the node is equipped with sensors for temperature, light and humidity

measurements, and allows to attach other devices through two expansion connectors. From the firmware point of view, the system runs TinyOS, a light yet efficient open source operative system, developed at University of California, Berkeley, USA. **TelosB** [31] is the same design.

MICA2 [32] and **MICAz** [33] are slightly more computationally powerful then TelosB, at the price of a higher power consumption. An Atmel ATmega128L [34] represents the core of the node: it is an 8-bit, RISC based MCU, clocked at up to 8 MHz, and draws no more than 20 mA in active mode. It supports six different sleep modes, which allow to reduce power consumption down to a few micro-Watts, when the resources are not needed. MICA2 and MICAz provide respectively a 868/916 MHz multi-channel transceiver (the Chipcon CC1000), and an IEEE 802.15.4 compliant RF one (the Chipcon CC2420), but do not have any sensor already present on board; access to external devices is achieved through two expansion connectors. These nodes run a TinyOS based operative system called MoteWorks.

iMote2 [35, 36] is well above the previous nodes in terms of processing capabilities thanks to a high performance Marvell PXA271 XScale processor [37], a highly configurable processor supporting an operating frequency in the $13 - 413$ MHz range. It is also equipped with 250 kB of SRAM, 32 MB of SDRAM and 32 MB of flash memory, and has a built in Digital Signal Co-Processor with MMX instruction set for multimedia applications enhancement [38]. Power consumption in active mode ranges from 40 mW to several hundreds of milli-watts, depending on the clocking frequency. Low power modes are also supported, consuming as little as a few milliwatts in Idle mode, with the core supply voltage enabled, or even less than 1 mW in deep sleep mode, with the core completely off. The PXA27x XScale processor family was originally designed by Intel. The node provides several options for I/O interfacing, such as UART, I^2C, USB, camera



(a) Panoptes (b) Cyclops

Figure 2.2: Examples of WCN nodes present in the literature

interface, etc. A ChipCon CC2420 [25] implements the wireless interface through the IEEE 802.15.4 standard. iMote2 can run TinyOS, embedded Linux or University of California, Los Angeles' SOS [39].

We would like to point out that the presented current consumptions have been extracted from the datasheets of the chips. Measurement setups are not known, except in some documents in which the manufacturer declares that "all current consumption measurements are performed with all I/O pins configured as inputs and with internal pull-ups enabled" [34]. As we will see in Sec. 4.3, driving a pin of a peripheral requires lots of energy, therefore the provided currents have to be considered as lower limits.

2.2 Related Work

2.2.1 Panoptes

The first significant example of a WCN node is Panoptes [40, 4], first developed in 2003 by Feng *et al.* at the OGI School of Science and Engineering at Oregon Health & Science University and then improved at the Department of Computer Science of Portland State University. The first prototype is built on the Applied Data Bitsy board, based on the Intel StrongARM 206

MHz embedded processor [41], equipped with a Logitech 3000 USB-based video camera, which applies some proprietary compression algorithm on the acquired images, 64 Mbytes of memory, and an 802.11-based networking card. The system runs the Linux 2.4.19 Operating System (OS) kernel. The entire system consumes approximately 5.5 W while capturing, compressing and delivering the stream of images acquired at $18 - 20$ frames per second (fps) with a resolution of 320×240.

The authors moved to the Crossbow Stargate Platform [42] for the second prototype. The Stargate Platform features a 400MHz, Intel PXA255 Processor, supported by an Intel StrongARM SA1111, 64 MB of RAM and 32 MB of flash memory, from which the Linux 2.4.19 OS is loaded at start-up. A daughter board mounted on the processor board provides USB connectivity for the camera. In addition to being more powerful in terms of processing capabilities with respect to the Bitsy Board, the Crossbow Stargate Platform is also less power demanding (about 4 W).

From the point of view of the firmware, the authors developed a unit for power management, and a set of blocks for capturing, filtering, compressing and transmitting the images. The power manager is in charge of determine the best acquisition and wireless transmission policy according to the energy available, in compliance with the application. Video capturing consists on decompressing the data coming from the USB interface to allow image manipulation. Then, images are filtered to remove redundant information, e.g. by detecting the only portions in the scene that has changed with respect to the previous frame. The filtered images are then compressed, e.g., with JPEG, to reduce the amount of power spent in the transmission, and then sent to the wireless transceiver, which is in charge of managing the available bandwidth by selecting the frames to the transmitted according to a priority-based mechanism. The nodes communicate to a host-PC in which a surveillance application merges the generated in-

formation for surveillance purposes.

The overall performance strongly depends on the specific implementation of the video capturing, filtering and compressing blocks. In particular, the amount of processing load is strictly connected to the resolution of the acquired images and the routines developed for image compression. The authors analyzed the power consumed by one board in several configurations (idle, camera on, camera and networking on, etc.), finding out that 1/3 of the power budget is spent for wireless transmission. In order to save power, the system switches components off or sets them to low power modes whenever possible. In general, this approach partially solves the problem, as on/off or wake/sleep transitions may require a substantial amount of energy.

2.2.2 Cyclops

Another important example of WCN is Cyclops [5], an electronic interface between a camera module and a standard WSN host mote. It was developed by Rahimi *et al.* at the Center for Embedded Networked Sensing, UCLA, Los Angeles, CA, USA in conjunction with the Agilent Technology Laboratories, in Palo Alto, CA, USA.

Cyclops consists on an Atmel ATmega128L MCU supported by a 64 KB SRAM and 640 KB of flash memory. The MCU, based on an 8 bit RISC architecture, is clocked at approximately 8 MHz and interfaces with a 352×288 CMOS imager through a CPLD, which is also responsible for simple but fast image processing while receiving data. The combination of CMOS imager, an ADCM-1700 from Agilent Technology [43], and a CPLD, a Xilinx XC2C256 CoolRunner[44], brings several advantages. CMOS imagers are cheap and low-power, and at the same time they allow to perform some control and processing directly on-chip. CPLDs are fast, overcoming the limitations of low-power MCU for data exchange and processing at high

speeds, and power efficient, since negligible energy is consumed when their clock source is disabled. This can be done as soon as they terminate their tasks, without the need to turn them off completely. The MCU represents the core of Cyclops: it is responsible for the management of the entire node, selecting the power mode of the other elements, determining what and when to transmit, and interfacing with the host mote. The CPLD is used as interface between the sensor and the MCU itself, which is too slow to cope with the sensor output data rate, and as a first level image processor that operates while grabbing the image.

The firmware in Cyclops consists of a set of drivers required by the MCU to communicate with the peripherals, a library to perform image manipulation both at low (e.g., matrix operations) and high level (e.g., background subtraction), and a so called "Sensing Application" which receives commands from a host-PC and executes them, exploiting the available resources. Tested applications include a simple object detection algorithm, based on background subtraction, and hand postures recognition for human-computer interaction.

The joint use of a low-power device and of a fast one achieves a significant (one order of magnitude) improvement in terms of energy requirements with respect to Panoptes (the authors declare a power consumption of less than 100 mW for Cyclops, to be added to the consumption of the external sensor network host node), but it offers limited processing capabilities and a low acquisition rate (only 5 fps, according to the imager's datasheet).

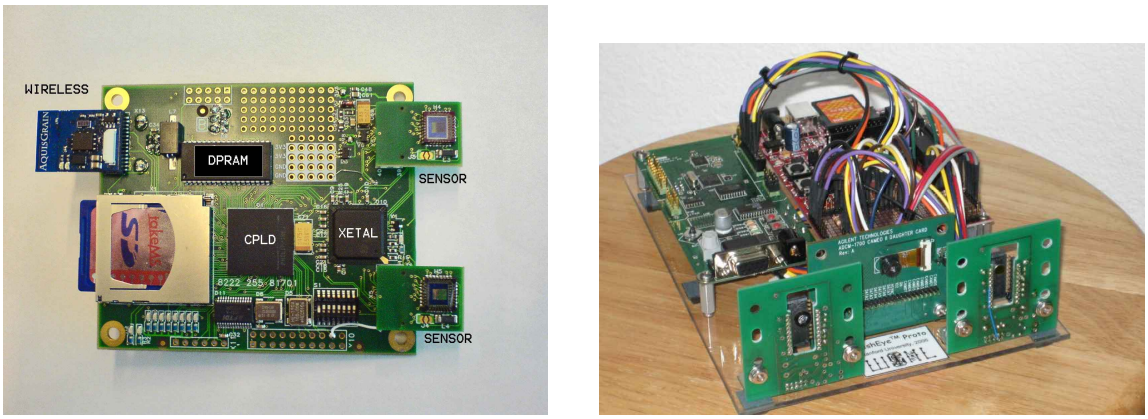
2.2.3 Philips Smart Camera Mote

Kleihorst *et al.*, from the Philips Research Laboratories, Eindhoven, The Netherlands, proposed a different solution in [6]. In order to minimize the time spent for pixel-level processing on the images provided by up

to two VGA color image sensors, the node exploits a Massively-Parallel Single-Instruction, Multiple-Data (MP-SIMD) processor, the Xetal-II [45]. The architecture of this processor consists on a linear processing array with 320 processing elements supported by a 10 Mb on-chip memory to buffer the image. Three further processor are present, managing the input stream, the output stream and for global control respectively. Everything is included in one sigle chip, which is clocked at 84 MHz. The rationale behind this approach is that often, when processing an image, the same calculations (e.g., convolution) are repeated on relatively small blocks of neighboring pixels by sliding a "kernel" throughout the image. The SIMD architecture exploits such intrinsic parallelism of images by elaborating all the fetched data in parallel with the same instruction, thus providing a very high throughput and low latency even with high resolution images and complex image-processing algorithms. As an example, the authors tested the processor with a 11×11 convolution on a 640×480 image, requiring as little as 2 ms to execute. Power efficiency is achieved also, since the mechanism reduces the number of accesses to the memory and cuts down the overhead for control and address decoding. Nevertheless, the absolute power consumption is high, with a peak value on the order of several hundreds of milli-watts. With such requirements, it would not be feasible for such a node to run on batteries continuously, acquiring several frames per second. On the other side, the Xetal-II power efficiency perfectly suits to camera-based WSNs, enabling many applications which usually require a pc connected to the power outlet to run. In a multi-tier network of wireless nodes, this node would lie near the highest tier.

2.2.4 MeshEye

Sensor diversity is the main characteristics of MeshEye [7], developed at the Wireless Sensor Networks Lab of Stanford University, CA, USA by



(a) Philips's smart camera node (b) MeshEye

Figure 2.3: More examples of WCN nodes present in the literature

Hengstler and Aghajan. This node hosts up to eight (but two are actually used) low-power, low resolution, low color depth imagers and one VGA camera. In particular, the ADNS-3060 optical mouse sensors [46] (3030 pixel, 6-bit grayscale), and the ADCM-2700 CMOS camera module [47](480640 pixel programmable, grayscale or 24-bit color) by Agilent Technologies are the ones present on the board. A high performance MCU, the Atmel AT91SAM7S64 [48], represents the node core and is responsible for both control and data processing. It incorporates an ARM7TDMI ARM Thumb processor based on a power-efficient 32-bit RISC architecture that can be clocked up to 55 MHz, a 16 KB SRAM and a 64 KB flash memory, and is supported by an external MMC/SD 32 MB flash memory card for temporary frame buffering/image archival. The acquisition policy consists on performing acquisitions having only one low-power imager on, to perform basic motion detection, when the scene is static. If the node detects an event, it turns on also the second low-power imager to perform stereo-vision based 3D blob dimensioning and localization, and a high resolution image is taken with the VGA camera. The output of the two low-power imagers is used to determine the portion of the high resolution image in which the detected object is present, in order to avoid transmission of uninformative

data. Then, the ChipCon CC2420 2.4 GHz IEEE 802.15.4/ZigBee-ready RF transceiver allows wireless communication.

The authors claim that, when the frequency of events is low, the lifetime of the node, powered with two AA batteries (2850 mAh), ranges between ten and forty days, for an average current of 3 mA in the best case. However, the frame rate of the system is low (less than 2 fps]); increasing the acquisition rate significantly reduces the lifetime of the node, with an increment in power consumption which is almost linear. Nevertheless, a frame rate of at least 10 fps is often desirable, but such a high acquisition rate would deplete the batteries in less than two days.

2.2.5 Yale's AER imager-based node

All of the systems above acquire and process images at full resolution and full pixel depth. This approach requires substantial memory to store the images and powerful processing units to process them. A different approach was taken by Teixeira *et al.* [9] at Yale University, New Haven, CT, USA. Their work proposes a non-standard imager in which the concept of frame is replaced by an Address Event Representation (AER). In practice, every pixel is uniquely identified by an address and, instead of just measuring the amount of light impinging on each pixel, it senses a precise property of the scene, such as temporal and spatial difference. Every time the measured "amount" of such property exceeds a predefined threshold, the pixel address is "fired". The higher the frequency of firings associated to a pixel, the greater the intensity of the phenomenon for that pixel. No image is actually generated, at least not in the way we are used to: the concept of frame is totally missing and the generated data are not directly readable by humans. Nevertheless, this approach simplifies the detection of the portions of the image in which most of the activity is taking place and might get closer to the way a machine thinks. In fact, such architecture moves a

significant amount of processing inside the sensor, therefore reducing the overall processing and power requirements. As it will be discussed later in Sec. 3.1.2, this is a crucial point in the development of low-power WCN nodes.

With the data generated by this sensor, the authors were able to develop applications for assisted living, including recognition of behaviors of people in a house. Unfortunately, the sensor node is a standard, high-performance iMote2, featuring an Intel XScale processor, operating at more than 100 MHz, supported by a 32 MB SDRAM and a 32 MB flash memory. Such high processing capabilities have a critical impact on the power requirements, which are in the order of hundreds of milli-watts. Moreover, the previously described camera has not been really mounted on the mote, but just simulated. In the end, it seems that no measurements on a real prototype have been carried out to prove the effectiveness of the approach.

2.2.6 Other nodes

The previous list of research works does not include many other nodes that have been proposed in this field. Our selection of works is focused on power aspects rather then application development, networking or any other issue. Here in the following we will briefly introduce other popular nodes which, in our opinion, brought a marginal contribution, but still important, in our field of investigation.

Cao *et al.* proposed a mote in [49] in which image acquisition, image processing and data compression are delegated to an FPGA, while sensor control and RF module management are performed by an embedded 32-bit, RISC processor by Samsung, the S3C44B0X [50]. An external flash mem-

ory and SDRAM are required to store the program and for data buffering. As RF module, the ChipCon CC1000 has been chosen. Images are processed with the aim of detecting unusual events, in which case the acquired frame is compressed and transmitted. No power consumption analysis has been provided, but according to the hardware equipment we can expect it to be in the order of the watt.

WiSN nodes [51], where the "i" stands for "image", presented by Downes, Rad and Aghajan from the Wireless Sensor Networks Lab of Stanford University (CA, USA) are the very first example of nodes equipped with heterogeneous imagers, and basically represent the embryonic form of MeshEye, since they were developed in the same laboratory and are based on the same processor (please refer to Sec. 2.2.4).

eCam [52] is an "ultra-compact, high data-rate wireless sensor node with a miniature camera". Basically, it consists on a camera module integrating a multi-resolution camera, the OmniVision OV7640 [53], attached to the Eco Wireless Sensor platform [54], a tiny ($\simeq 1$ cm^3) host-node, integrating a 2.4GHz RF transceiver with embedded 8051-compatible MCU[2] and ADC, the NRF24E1 by Nordic Semiconductor [56], and 3-axial acceleration, temperature and optical sensors. The node can interface with other modules via a 16 pin expansion connector. The system processing capabilities are very limited, no image processing is performed. All the node can do is acquiring images and transmitting them. The low power characteristics of the node do not seem to fit to camera-based sensor networks.

CMUcam3 [57] is mainly oriented towards a simple programming interface, thanks to its open source framework and C programming. It consists on an Omnivision camera module, either an OV6620 [58] or an OV7620 [59],

[2] "The 8051 is an 8 bit MCU originally developed by Intel in 1980. It is the world's most popular microcontroller core, made by many independent manufacturers." [55]

a NXP LPC2106 microcontroller [60], based on a 32 bit ARM7TDMI-S processor, and a Fitst In, First Out (FIFO) memory chip for image buffering, an Averlogic AL4V8M440 [61]. The presence of the FIFO significantly reduces the complexity of the system, but at the same time introduces an important source of power consumption, which is in the order of several hundreds of milli-watts for the entire node. CMUcam3 has been interfaced with a Firefly platform [62] in [63] to achieve a complete WCN system. The Firefly platform is a low-power custom node developed at Carnegie Mellon University, Pittsburgh, PA, USA, which achieves a tight time synchronization among nodes.

MicrelEye [64] consists of a hybrid architecture System on a Chip (SoC), the Atmel FPSLIC [65], which contains a low power MCU and an FPGA. A CMOS sensor, 1MB of external SRAM and a Bluetooth module complete the mote hardware configuration. The tasks delegated to the MCU include sensor configuration and part of the object recognition algorithm, while the FPGA performs image capturing, SRAM access and Bluetooth management, image processing and the top level control through a finite state machine. As software application, an Support Vector Machine-based object-detector has been developed. Power consumption is in the order of hundreds of milli-watts.

Citric [8] is a daughter board of a Tmote Sky host-node. It integrates an OmniVision OV9655 [66], a low power, multi-resolution (from 40×30, up to 1280×1024 pixels) CMOS camera module with image processing capabilities, and a Marvell PXA270 embedded processor connected to a 64 MB SDRAM and a 16 MB FLASH for image buffering and code storage respectively. A microphone is present too. Developed applications include image compression, target tracking and camera localization. Due to the very high processing capabilities, power consumption is almost 1 W.

In this section we have listed the WCN nodes proposed by researchers in the past years: most of them offer good processing power, but they run algorithms on images provided by high resolution, high color depth sensors. Therefore, the latency delays introduced by image processing and the power consumed by the entire node are often so high that the "low power" characteristics are achieved only by reducing the frame rate to one or two frames per second.

This general condition conflicts with the requirements set by applications in the fields of security, automotive, animal monitoring, etc., where the dynamics of the monitored objects is much faster than a second. Consider for example an application in which we want to monitor the speed of cars on a highway. At the speed of 130 km/h, cars cover more than 36 meters in a second. Many frames are required in order to provide a reliable measure. At low frame rates, we would need to place the node far away from the scene to have a view wide enough to take several snapshots of the same object with the same camera. This may not be feasible and in general is not desirable.

We think that most of the previously described nodes would fit throughout the tiers of a multi-tier WCN, but none of them could occupy the lowest tier, meaning that the whole network would not be able to survive for a long period of time. Even MeshEye nodes (presented in Sec. 2.2.4), which try to overcome the problem exploiting both low- and high-resolution imagers, do not succeed in achieving a long lifetime. In fact, the presence of a high performance MCU, which is required to process the high-resolution images, reduces the advantages brought by the presence of low-resolution cameras.

There is the need for ultra-low-power nodes that monitor the scene continuously at high frame rate, even in a coarse way, looking for events of interest to happen, and that generate wake-up triggers for higher tiers only

in particular circumstances. This would guarantee a better usage of the energy resources, increasing the network lifetime by one, even two orders of magnitude. Moreover, it would allow even to more powerful nodes to take part to the system. Within the previous example, the task of the nodes belonging to the lowest tier would consist on detecting the presence of cars and possibly determining a rough estimate of their speed. They would then activate the higher tiers only when they discover the presence of a fast car. In practice, they would act as a filter that cancels all the events that are clearly unimportant.

Chapter 3

Designing an Ultra-Low-Power WCN Node

There are several problems underlying the design process of an ultra-low-power WCN node. We will analyze these problems and suggest a design method that leads to a node that can work for months powered with a couple of standard batteries. This chapter is organized as follows. In Sec. 3.1 we will identify the logic elements that make up a WCN node and we will summarize the characteristics that the hardware embodiments of such elements need to have in order to achieve ultra-low power consumption. In Sec. 3.2 we will discuss about the firmware implementation, describing how to manage the hardware resources to get the most out of our node, how to exploit the low-power characteristics of the devices and when to use the high-processing capabilities present on board. Finally, the power model corresponding to such a design is described in Sec. 3.3.

3.1 Hardware

Within an embedded system we can always identify a set of *logic elements* that make up the system, i.e., a set of entities with specific tasks to carry out, receiving commands and data as inputs, and generating outputs that

will be used by some other entities. Each logic element provides its contribution to accomplish the duty for which the entire system has been designed for. This can be also seen in a hierarchical way, with logic elements that are divided into smaller elements.

Each logic element has an *hardware embodiment* in which it is implemented, a device that has been designed or programmed to carry out the tasks assigned to the logic element itself. The association between logic elements and hardware embodiment is not necessarily bijective: the implementation of a logic element could be split into several devices and, at the same time, one single device may contain more than one logic element.

We conceive a WCN node as made up by four main logic components:

- a *control unit (CU)*, which manages the whole node, deciding when the other logic components need to be operative and coordinating their activities, aiming at executing the node's tasks spending the minimum amount of power;
- a *sensor*, seen as an entity that transforms the information present in the outside physical world into data that can be processed by a computing system;
- a *memory*, that acts both as a buffer where to store the data coming from the sensor, waiting to be processed, and as a place where to store the processing results, which may be needed in the future;
- a *processing unit (PU)*, which runs algorithms on the acquired images, trying to reduce the amount of data to transmit as much as possible;
- a *transceiver (TRX)*, for wireless communication.

We will try now to identify the most suitable hardware embodiment for each of these logic components according to their tasks.

3.1.1 Control unit

The CU's job consists of several simple tasks such as providing timing signals to the sensor, managing the data transfer from the sensor itself and the memory, activating/disabling the PU, deciding when to enable wireless transmission and reception. Such tasks do not typically require high timing resolution and no high processing capabilities are required too, as long as most of the processing is carried out by the PU. Therefore, the CU can be implemented on an ultra-low-power device, clocked at a very low frequency (low frequency clocks easily exceed the kilohertz, thus achieving a precision smaller than 1 μs, which is sufficient for most sensors). This is the case, for example, of low-power microcontrollers, as the Texas Instruments MSP430 Microcontroller, which is also present in the Tmote Sky and the TelosB nodes (refer to Sec. 2.1.4). The availability of a well supported embedded OS such TinyOS is also an important factor, since it provides facilities for time synchronization among nodes, multi-hop routing, self-management of the network, etc. An ultra-low-power PLD represents a second option: it allows to design multiple Finite State Machines (FSMs) operating in parallel, to disable portions of the system by simply AND-gating their clock and to employ multiple clock domains. When needed, it can also generate high accuracy timing signals with a high frequency clock that is fed to a small portion of the internal logic, with a limited effect on the overall power consumption.

3.1.2 Imager

The choice of the imager has a strong impact on power consumption. The higher the resolution and the color depth, the higher is the amount of data to transfer, buffer, and process. The activity on input and output pins deeply affects consumption, and its effect is multiplied by several times

if the hardware unit responsible for processing does not provide enough memory to buffer the entire image. In this case, another device such as a FIFO memory has to be involved in the entire process. Therefore, the designer should select the imager that provides enough, but not more visual information than necessary to run the algorithms.

Although standard imagers are mere light sensors, it may be convenient to use a device that senses specific visual properties of the scene, such as local contrast, image texture, or motion. We will call such device a *"vision sensor"*, as opposite to image sensors. This approach can reduce the power consumption of the overall node. First of all, the chip generates less data, thus limiting signal activity and required memory size. Performing operations on-chip rather than on an external component is also power-efficient. Processing is inherently parallel, with the same local operator replicated at each pixel. By performing the initial (and often most computationally intense) operations at the sensor level, subsequent (and more power hungry) processors only need to deal with selected frames and image areas, with positive impact in terms of latency, device occupancy and system complexity. For instance, suppose that the application requires to enhance the edges of the acquired image. Assuming a resolution of 128×64 with 8 bpp, we have to transfer and buffer 64 Kb of data. Then, edges can be extracted sliding a 3×3 mask throughout the image. This operation requires, for each pixel, 9 multiplications, 8 additions and 1 division. Thresholding may be used to reduce the amount of data down to 8 Kb, too. Performing such operations on a processor running at 10 MHz, assuming only 1 clock cycle per operation, introduces a latency of 15 ms. Clearly, such memory and processing requirements are not suitable for an ultra-low-power system. On the other side, a custom imager can achieve the same goal by performing the processing task in the analog domain or in the in-pixel digital circuitry, during the acquisition. The amount of data to be transferred and buffered

is reduced by a factor of 8, no latency is introduced and the requirements in terms of processing capabilities can be relaxed. This comes at the expense of a more complex sensor design process.

If most of the initial processing is performed at the sensor level, the sensor itself can communicate whether a frame contains relevant image data to be further processed. For example, the sensor may produce the number of "active" (interesting) pixels in a frame; accordingly, the CU may then decide whether the image data should be further analyzed by the PU or not. In the first case, we will say that the sensor is in *Active mode*, which implies data transfer to the PU. In the second case, the sensor is in *Idle mode*, and no data needs to be transferred. Note that, even when in Idle mode, the sensor still acquires and processes the image, although data is not transferred to the PU.

In other words, vision sensors merge the sensing unit with part of the processing unit, moving portions of the whole computation closer to where data are generated. This turns to be very efficient especially in case of mask filters, that operate on small groups of neighboring pixels and thus allow to exploit the spatial proximity of the sensor's photodiodes. This helps to save time and energy with respect to a processor which needs to fetch data located far away in the memory.

3.1.3 Memory

A memory unit is required every time the processing unit is not able to process the data right away as they become available at the sensor output pins. This situation occurs most of the times, e.g., when we have no complete control over the readout process, as in the case in which the sensor provides output data in a single burst after a "start" trigger. Or we may need to compare two images, for motion detection or background subtraction. We may also need to run more than one image processing

algorithm on the same image. Again, we may decide to apply a filter on the acquired image only in particular circumstances, according to some parameter extracted from the image itself.

Mainly, the memory component must satisfy the requirements set by the node in terms of size, speed and power consumption. As usual, these parameters are conflicting. It is also important that Direct Memory Access is supported when the burst of data runs at high speed. Other important parameters include latency and interface modality.

In the light of these consideration, Static RAMs (SRAMs) seem to be the right solution, since they are faster and consume less power with respect to Dynamic RAMs (DRAMs), which also require data refresh. The problem with SRAMs lies in their price, which is much higher due to the higher number of transistors required per memory element, if compared to DRAMs. Another possibility involves the use of soft memories implemented in a PLD, which, by the way, are usually built on SRAM technology. This solution allows an easy interface and typically satisfies the requirements in terms of speed imposed by the sensor. This is optimal when the CU and/or the PU are implemented on the same PLD, since system complexity is reduced. In addition, such an architecture allows to reduce power consumption simply by disabling the soft-memory clock.

3.1.4 Processing unit

Unlike in standard WSNs, where the major source of power consumption is represented by the only transceiver [67], in camera-based SNs the contribution brought by the PU becomes as much as important. In fact, WCNs combine the huge amount of data generated by the sensing unit with the high acquisition rates imposed by the applications. Thus, the PU is responsible of reducing the entire stream of data to a single, short message that can be transmitted wirelessly spending a limited amount of energy,

Moreover, this "information synthesis" task needs to be carried out in a limited amount of time. Therefore, once the right sensor has been identified, the designer must choose the processing device that allows to meet the two conflicting requirements set by the application: processing power and power consumption.

The required processing capabilities depend on the complexity of the algorithms to implement and the rate at which such algorithms are executed. Basically, the time t_{proc} required for processing has to be much shorter than the frame period t_{frame}. We say "much shorter" because we would like to execute the algorithm in a fraction of the frame period, so that the PU stays idle for most of the time. Once the algorithm has been designed, and given the resolution $(m \times n)$ of the images provided by the sensor, the value for t_{proc} depends on the clock frequency f_{clock} and the architecture of the PU itself. In fact, the same instruction can be executed in one clock cycle by a processor and in several clock cycles by another processor. We can measure the efficiency of a processor through the average number of Instructions Per Cycle (IPC) that it performs while executing the algorithm. Features such as hardware multipliers, availability of floating point arithmetics and multitasking affect such parameter. From our point of view, in order to take into account the complexity of the algorithm, we extract the average number of clock cycles per pixel N_{cpp} required to process the images, which is given by the IPC multiplied by the number of instructions per pixel. In the end, it has to be that:

$$t_{proc} = \frac{(m \times n) \times N_{cpp}}{f_{clock}} \ll t_{frame}. \tag{3.1}$$

For instance, suppose we equip our node with a 128×64 camera that acquires images at up to 30 fps, and suppose that the algorithm that we need to run requires roughly 100 clock cycles per pixel to generate the

output. Well, in this case, we would need a processor that can run at:

$$f_{clock} \gg \frac{(m \times n) \times N_{cpp}}{t_{frame}} \simeq 24 \text{ MHz}. \tag{3.2}$$

By employing FPGAs instead of processors, we can significantly relax the requirements. This is due to their intrinsic ability to process data in a parallel and pipelined fashion, which allows to reduce the N_{cpp} parameter by several factors.

Within the given set of devices which fulfill the requirements in terms of computational capabilities, we need to find the one(s) that satisfies the power requirement set by the application. Generally speaking, there are two working modes that we need to take into account. The node can either monitor the scene just looking for some event of interest to happen or carefully analyze the full image when a potential event has been discovered. In the former case, we will apply simple and fast algorithms, so that the PU spends most of the time in a low-power mode. In the latter case, we need to produce accurate results about the ongoing events, therefore we would like to exploit the PU at its full potential. For each of these cases, we define a selection criterion that applies to the power consumed in low-power and active mode respectively, whose values are present on the datasheet of each device. The constraints are given by the capacity C provided by the node's power source and the minimum expected lifetime L.

Let's first consider the low-power working mode and determine the maximum current $i_{low-power}$ that is allowed by the constraints. Since the entire network will succeed in achieving long lifetime only if the PU operates in this mode in the great majority of the cases, and since we are just defining "order-of-magnitude" requirements, we can assume that the node always operates in low power mode. Thus, it has to be that:

$$\frac{C}{i_{low-power}} \lll L. \tag{3.3}$$

For the maximum current i_{active} that the device can draw in active mode, we need to change our assumptions. Let's consider the ideal case in which no power is wasted in idle mode, i.e., $i_{low-power} = 0$; then it has to be that:

$$\frac{C}{R \times i_{active}} \ggg L, \tag{3.4}$$

where R denotes the percentage of time spent by the PU being fully operative. Such ratio can be estimated according to the application and the complexity of the algorithms we are going to implement.

The "$\ggg$" relations, meaning "one order of magnitude greater than", are required to compensate the approximations, above all the fact that we are assuming to devote all the energy to the PU.

Just to provide a numerical example, consider $C = 2200$ mAh and an expected lifetime of a month, i.e., $L = 720$ hours. For the current drawn in low power mode, according to Eq. (3.3) we get:

$$i_{low-power} \lll \frac{C}{L} = \frac{2200 \times 10^{-3}}{720} = 3 \text{ mA}, \tag{3.5}$$

and thus:

$$i_{low-power} \sim 100 \ \mu\text{A} \tag{3.6}$$

Assuming that the time spent by the PU in active mode is 1‰ of the total time (meaning that, on average, we detect one event per hour, and that the event lasts four seconds), from Eq. (3.4) we obtain:

$$i_{active} \lll \frac{C}{L \times R} = \frac{2200 \times 10^{-3}}{720 \times 10^{-3}} = 3 \text{ A}, \tag{3.7}$$

therefore:

$$i_{active} \sim 100 \text{ mA}. \tag{3.8}$$

Of course, these are rough estimates which the designer needs to carry out before choosing the hardware embodiment for the PU. There are other features which we may look for in a device. For example, support for

standard communication protocols (e.g. for data transfer from/to an external memory) and image processing libraries is also desirable, for easy programming.

3.1.5 Transceiver

In the context of WCNs, one may need to transmit entire images or even video streams to a host pc, where images can be processed by a powerful computer or analyzed by a human operator. In this case, the best option is to use a high performance wireless transceiver, based on the WiFi protocol. But this is not the case for ultra-low-power nodes, which are designed to perform the detection of events in a completely autonomous way and spread short messages such as alarms. Only in this way it is possible to achieve a long lifetime. If a snapshot of the scene is required, the ultra-low-power node may wake up a more powerful node that takes care of acquiring and transmitting the high resolution image.

Due to these considerations and for what has been discussed in Sec. 2.1.3, two main options for low-power communication are currently available: Bluetooth and ZigBee, with the latter being more suited for our case. We can safely say this because of Bluetooth-based devices typically provide higher data rates at the price of a higher power consumption. Even more importantly, the network topology supported by ZigBee introduces more flexibility with respect to Bluetooth, which is based on a Master-Slave scheme. ZigBee networks are self organizing, meaning that when we introduce a new node in the network, the process of node insertion is automatic, and we do not need to care about it. In Bluetooth networks such a process is not as easy, due to some constraints imposed by the network architecture. For example, one device can belong to several piconets, but can be master only in one of them. Again, each piconet has a 3-bit address space, meaning that the maximum number of devices that can take part of the

piconet is eight; when the ninth device asks to be part of the network, a new piconet has to be created, thus requiring a network re-organization. In the end, ChipCon's ZigBee-based transceivers seem to be the devices that best fit our case.

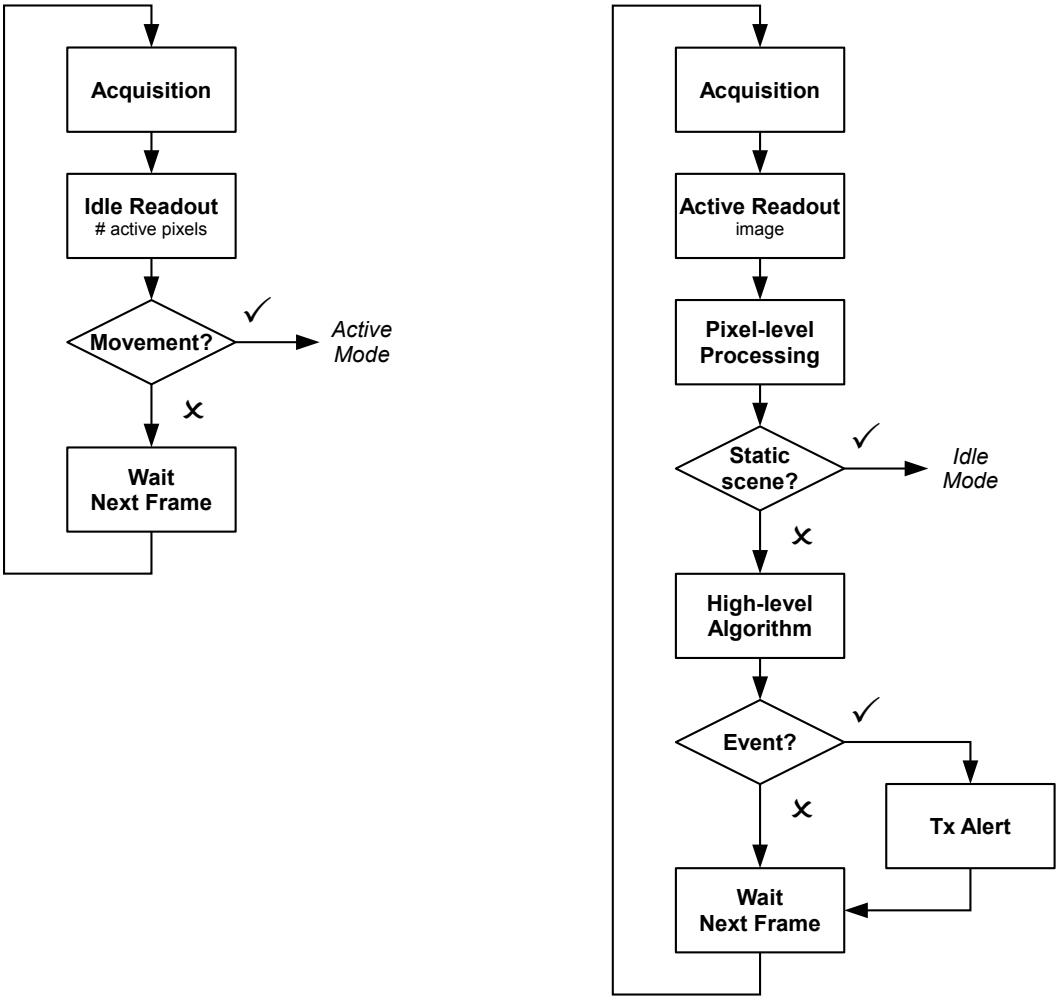
3.2 Firmware

By firmware we refer to the implementation of both the CU and the PU. The way the node controller and the image processing algorithms are implemented depend on the devices chosen to be part of the node's hardware. Of course, high-level programming languages provide easy and fast programming, while HDL such as VHDL allow to achieve a better performing system. Nevertheless, the functionalities should be the same, both if we use a MCU and if we use an FPGA.

The CU's main task is to ensure correct functionality of the node while maximizing its lifetime. In other words, the CU needs to create a time-base, in which each of the other logic elements of the system have to be mapped either as active or as idle throughout the frame period, according to what is happening in the monitored scene.

If we equip our node with a "smart" sensor capable of detecting events autonomously, as the one depicted in Sec. 3.1.2, we can consider two main operating conditions of the node. In the first one the sensor reports no "activity" of interest in the scene, meaning that the frame needs no further processing. We will refer to this operating modes as *Idle*, since a vast portion of the node's hardware will not be activated at all. In the second case, something is happening in the scene and the frame needs to be analyzed by the PU, or transmitted by the transceiver; therefore we will say that the node is *Active*.

While the node is in Idle mode, the CU only provides the timing signals



(a) Idle Mode

(b) Active Mode

Figure 3.1: Control Unit: flowcharts. In Idle mode, the CU provides the timing signals to the sensor, which generates as output a single value representing the number of non-zero pixels in the output image. The CU compares this value with a pre-defined threshold and selects the operating mode for the following frame. This is shown in (a). When something is happening in the scene, Active mode is enabled, and, after the acquisition, the entire image is read out. The processing unit operates at two levels: it first extracts some features from the image and, if things have changed with respect to the previous frame, it executes a high-level, decision making algorithm. If an event occurred, a message is transmitted to the other node. When the scene does not change for a relatively long period of time, the node is switched back to Idle mode.

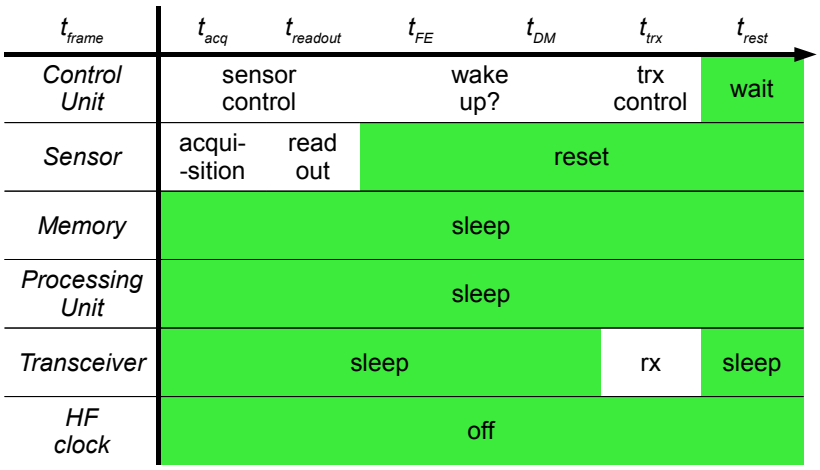


Figure 3.2: *Idle* frame period: status of the node's elements. The figure shows which logic elements are active and when within a frame period. Time flows from the left to the right. In Idle mode, the sensor monitors the scene autonomously, by generating as output only the number of non-zero pixels of the output image. The control unit compares this value with a threshold and, in case, switches the node to Active mode. Finally, it asks the transceiver to communicate with the other nodes. The other elements of the node, the most power hungry ones, are never activated, thus consuming very little power.

to the imager and asks it to count the number of active pixels present in the acquired frames, i.e., the imager itself is set to operate in Idle mode. At each frame, the CU evaluates the reported number of active pixels and decides either to remain in Idle mode or to switch to Active mode. The flowchart in Fig. 3.1(a) summarizes this process. Moreover, the CU disables the memory and the PU, while the communication activity is limited to receiving information from other nodes. In this case, since the only CU needs to be operative, and since it has to be operative all the time, the system should be clocked at a low frequency, thus achieving power saving. This is shown in Fig. 3.2, where the status of each element in the node throughout the frame period is shown.

The frame period is divided into six intervals: t_{acq} represents the imager's integration time, after which the sensor counts the number of active

pixels and provides the result at the CU. This is carried out during $t_{readout}$. t_{FE} and t_{DM} refer to the processing phase, which in this case is not performed, being the node idle. Then, the node communicates with the other nodes during t_{trx} before resting until the next frame. From Fig. 3.2, we can easily see that most of the node is sleeping, and minimum power is consumed.

If the number of active pixels exceeds a fixed threshold n_{pix}^{th}, the imager and the node are set to Active mode. When in this mode, after acquisition, the CU wakes up the memory and asks the sensor to readout the whole data. Once the readout has been completed, the CU activates the PU, which reads the data from the memory and processes it. A high frequency (HF) clock is usually required for this latter task, thus introducing an important source of power consumption. The minimum frequency at which such clock has to run has to be extracted from the algorithm complexity and the frame period. Of course, the criterion is that the processing phase must be short enough to allow the entire process to terminate without introducing delays for the following frame.

The PU typically performs a two-stage process: image Feature Extraction (FE), followed by higher-level Decision Making (DM). FE aims at extracting from the images the information that we care about, filtering all the useless data and synthesizing the information in a few concise parameters which will pass to the next phase. Sample FE algorithms include background subtraction, motion detection, edge detection, histograms of oriented gradients, etc. DM receives as inputs these features and generates the final output of the entire algorithm. At this stage of the process, we are not dealing with images anymore, but we are working at a higher level with the parameters of interest extracted from the images.

For example, the DM stage may consist in a classifier based on the features computed in the FE stage, as it happens in [64], where the authors

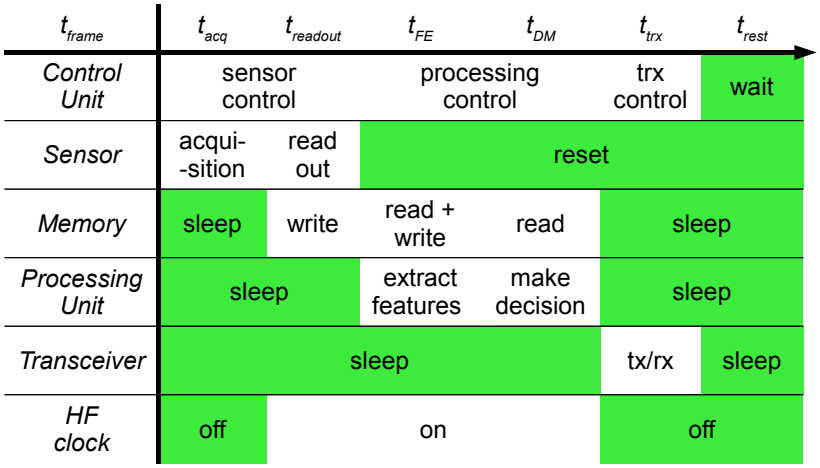


Figure 3.3: *Active* frame period: status of the node's elements. The figure shows which logic elements are active and when within a frame period. Time flows from the left to the right. In Active mode, after acquisition, the imager provides the entire image at the output pins. Therefore, the CU wakes up the memory to receive the data. Then, data are processed by the processing unit before transmitting the result to the other nodes of the network. The high frequency clock is enabled for the memory and the processing unit for all the time required to receive and process data, thus introducing a significant source of power consumption.

present a Support Vector Machine-based object detection system. Here, FE consists in a three phase process: the PU first performs background subtraction, and in a subsequent phase detects a Region of Interest (RoI) in which the only intrusive object is present. Finally, it generates the features by calculating the average gray values for each column and row of the RoI. The features are then provided to the SVM, which represents the DM algorithm.

The activity of the node when active mode is enabled is shown in Fig. 3.3. We can easily see that power consumption increases significantly, due mainly to the activation of the memory, the PU, and the high frequency clock source. The memory needs to operate during the entire readout period $t_{readout}$, in which the image is written in the memory, and during the

FE and DM processing phases. While extracting the features (t_{FE}), the PU reads the image from the memory and then may need to write the results back to it. Then, decision making is performed during t_{DM}, and the PU reads from the memory the previously extracted features to generate the final output. The result is transmitted shortly after during the communication phase t_{trx}.

In general, the image features can be local, or a single feature may summarize the whole image content. In the latter case, we will say that the feature represents the *state* of the frame. Here is an example of a power-efficient structure for the PU. At each time instant, i.e., at each frame, the FE algorithm generates a state $s \in S$ from the images, where $S = \{s^1, s^2, \ldots, s^N\}$ and N is the cardinality of the state space. The DM algorithm keeps track of how the state evolves with time to generate its output y, expressed as $y = f(s_0, s_1, \ldots, s_t)$, where s_t represents a sequence of consecutive frames all with the same state. Notice that the output of DM stage may change only when a state transition occurs. Thus, at each frame, the PU needs to be activated for the amount of time necessary to accomplish the only FE stage if the state of the system does not change, or both the FE and the DM when the state changes.

After processing, several transmission policies can be considered. For example, one may transmit either full images, or simply the low-rate data produced by the image analysis algorithm. The latter approach is the preferred solution, since transmitting an entire image or even a portion of it would quickly deplete the batteries. If an alarm is sent and we really need to show to a human operator what is going on in the monitored scene or in case we need to analyze the scene more accurately, we had better employ our ultra-low-power node to wake up other, more powerful nodes, that can take care of the emergency situation.

Switching between Idle and Active modes needs careful planning in order

to maximize the node lifetime on the one hand, and reduce the risk of frequent switches from Active to Idle on the other, which would result in possibly missed detections of events and high energy wasted during transitions. A typical policy is to activate the node as soon as the measured amount of activity exceeds the threshold, in order to acquire as much information as possible about the event, and to switch the Active node back to Idle after a waiting period T_W in which the images are characterized by low activity (i.e., the number of active pixels in each frame is below the threshold). In this way we avoid the interruption of the detection algorithm while the event is still ongoing. For example, assume we perform motion detection in the FE phase and an object that we want to monitor enters the scene. As long as the object moves, we are able to track it, but if it stops for a few instants it disappears from the node's sight. Without applying the waiting period before switching the node to Idle mode, we would stop tracking the object. Of course we would start tracking it again when the object starts moving again, but in order to associate the previous event with this new one we would need a more complex system. Moreover, the system could keep on switching from one mode to the other with a negative impact on power consumption. By waiting for T_W, we introduce some flexibility in the system which allows us to avoid such problems.

3.3 Power Consumption Model

According to the structure described in Sec. 3.2, there are two main configurations in which the system can run:

(a) Idle mode, when no activity is detected for a period longer than T_W;

(b) Active mode, when something is happening the monitored scene and we need to process the images.

We will refer to the average power P_x consumed and the average time T_x spent by the system in these configurations with P_I, P_A and T_I, T_A respectively.

The P_x's vary according to the hardware configuration and the firmware implementation. On the one hand more powerful devices draw more current; on the other hand an efficient implementation of the CU runs at a low clocking frequency, while an efficient implementation of the PU terminates its task much earlier than the next acquisition starts, and can be disabled for most of the frame period. The values of the T_x's depend on the period of low activity that we wait before switching mode from Idle to Active, T_W, on the average duration of the events T_D, and the time between two following events T_E.

Typically, we expect events to be sparse in time, meaning that the period of events is much longer than their duration plus the waiting period, i.e.,:

$$T_E > T_D + T_W. \tag{3.9}$$

This basically means that events are far from overlap, and we expect the node to spend a certain amount of time in Idle mode before an event is detected. Therefore, when this condition takes place, in the interval of duration T_E that occurs between two following events, the node works in Active mode for a time T_D, during which the PU process the images. After the event has terminated, the system remains in Active mode and keeps monitoring the area for a T_W-long time window, just to be sure that the event effectively has terminated. Then, the CU activates the system Idle mode, which holds until a new event rises, i.e., after an interval of length T_I.

Conversely, if the frequency of the events is higher, meaning that:

$$T_E \leq T_D + T_W, \tag{3.10}$$

the sensor's Idle mode is never enabled.

We can summarize the two situations with:

$$T_I = max\{0,\ T_E - T_D - T_W\} \tag{3.11}$$

$$T_A = min\{T_E,\ T_D + T_W\} \tag{3.12}$$

Accordingly, the overall average power consumption turns out to be:

$$P = \frac{T_I \cdot P_I + T_A \cdot P_A}{T_E} \tag{3.13}$$

The impact of T_W on the lifetime depends on the ratio T_W/T_E, thus having its maximum for small values of T_E and decreasing accordingly. Therefore, by using greater values for T_W, we are safer since we reduce the probability of interrupting the process of event detection while it is still happening, but we pay a higher power consumption. The minimum value for T_W is given by the application, which is characterized by an average period of time in which objects temporarily stop their activity in the middle of an event. For example, consider we are tracking people in a room and we perform background subtraction in order to determine if an event is occurring; there may be some occluding objects which are part of the background such that the imager is not able to detect any activity. In this case we may choose T_W to be equal to the average time required for a person to walk through the hidden zone. This is usually limited to very few seconds. A safer approach consists on choosing T_W equal to the average duration of events T_D.

Using (3.13) and assuming to power the node with a battery of capacity C and voltage V, we can easily estimate the node's lifetime $T_{Lt} = C \cdot V/P$ as a function of T_E. In particular, (3.13) shows that the node lifetime increases as T_E increases, as expected.

Now, consider the power efficient implementation of the algorithm described in Sec. 3.2, based on a feature extraction phase that provides as output one among finite set of states. In this case, things change a little, since we have three main configurations in which the system can run:

(a) Idle mode;

(b) Active mode, when no system state transition occurs, therefore only the FE algorithm is executed;

(c) Active mode, when the system state changes, thus requiring the execution of the DM algorithm also.

We can replicate the same reasoning as above, thus we have P_I, $P_{A,FE}$, $P_{A,DM}$ and T_I, $T_{A,FE}$, $T_{A,DM}$ that represent respectively the power consumed and the time spent in each of these configurations. In addition, within the period of time in which an event takes place, we have the probability Pr_{trans} that a state transition occurs at each frame. Therefore, when we consider the situation represented by (3.9), in which the period of events is much longer than their duration, while an event is happening the PU always extracts the features (i.e., the state), but it executes the DM algorithm only once every Pr_{trans}^{-1} frames, on average. After the event has terminated, the system remains in Active mode and keeps monitoring the area for a T_W-long time window, in which the system state does not change, being the scene static, and DM is never performed.

The other situation needs to be split into two cases: if $T_D < T_E \leq T_D + T_W$, the sensor's Idle mode is never enabled, while the amount of time in which the scene is static is less than T_W. While if $T_E \leq T_D$, a state transition can happen at any times.

Thus we have:

$$T_I = max\{0,\ T_E - T_D - T_W\} \tag{3.14}$$

$$T_{A,FE} = min\{T_E,\ T_D\} \cdot (1 - Pr_{trans}) + \\ + min\{max\{0,\ T_E - T_D\},\ T_W\} \tag{3.15}$$

$$T_{A,DM} = min\{T_E,\ T_D\} \cdot Pr_{trans} \tag{3.16}$$

And the overall average power consumption is given by:

$$P = \frac{T_I \cdot P_I + T_{A,FE} \cdot P_{A,FE} + T_{A,DM} \cdot P_{A,DM}}{T_E}. \tag{3.17}$$

The dependence of $T_{A,DM}$ on Pr_{trans} underlines the need to developed as simple an algorithm as possible, i.e., an algorithm based on a FE phase in which the number of possible states that we assign to the system is low. By doing so, we reduce the probability of state transitions, lower the complexity of the PU and shorten the time required for processing. This translates into the HF clock running for a shorter period of time within the frame period, thus achieving lower values for $P_{A,FE}$ and $P_{A,DM}$.

Chapter 4

Implemented Node Architecture

In order to prove the effectiveness of our approach, we have developed a prototype based on an ultra-low-power sensor and a flash-based FPGA [68]. The sensor is a custom contrast-based vision sensor prototype designed for ultra-low-power applications. It automatically detects the presence of moving objects in the scene and wakes up the node when something relevant is taking place. The node then runs a custom people counting algorithm and provides as output just the number of counted people that cross the monitored area.

In the process of selecting the processing devices that best fit our case, we have been monitoring the market of low-power processing devices, taking into account all the possibilities as we pointed out in Sec. 2.1.2. Among the selection criteria we considered power consumption, processing capabilities, easiness of programming and impact on the node's complexity. We ended up choosing one single flash-based FPGA, which features ultra-low static power consumption. When clocked at a low frequency, current drain is very limited. Moreover, thanks to the availability of multiple clock domains, this solution allowed us to reduce system complexity by implementing both the CU and the PU on the same device. At the same time, PLDs allow to build efficient architectures for data processing. Of course,

we had to pay the price of a long and tedious programming phase, but the achieved results are worth the effort.

Despite being designed for wireless applications, our prototype does not include the wireless transceiver. The complexity that such component would add to the node design is significant and would have delayed the research activity. Nevertheless, the purpose of this thesis is to propose a method for designing ultra-low-power nodes based on a camera, which translates into reducing the power consumed by the entire node by one to two orders of magnitudes with respect of the state of the art. We have seen in Sec. 2.2 that the WCN nodes present in the literature exhibit a power consumption in the order of hundreds of milliwatts, most of which spent to process and/or transmit images. In our case, we concentrated the activity on developing an ultra-low-power control unit and an efficient algorithm that reduces the amount of data to transmit to a few bytes. By succeeding in doing this with much less than ten milliwatts, we can safely claim that we reached our goal. In fact, we believe that the presence of a wireless transceiver would not yield a dramatic increase in power consumption with respect to the values we measured. We could also consider our prototype as a daughter board to be attached to a standard WSN node such as Moteiv's TmoteSky or Memsic's TelosB. Having said this, in this chapter we will develop our considerations as if the transceiver was present, in order to provide a more complete view of the node.

In this chapter we will analyze in details the hardware and firmware architectures of our prototype, underlying how the system exploits the low-power characteristics of the devices. In Sec. 4.1 we define the node architecture from a hardware point of view, while the firmware is described in Sec. 4.2. We performed power measurements on the node and the results are shown in Sec. 4.3, followed by an estimate of the node lifetime when powered with a couple of standard batteries.

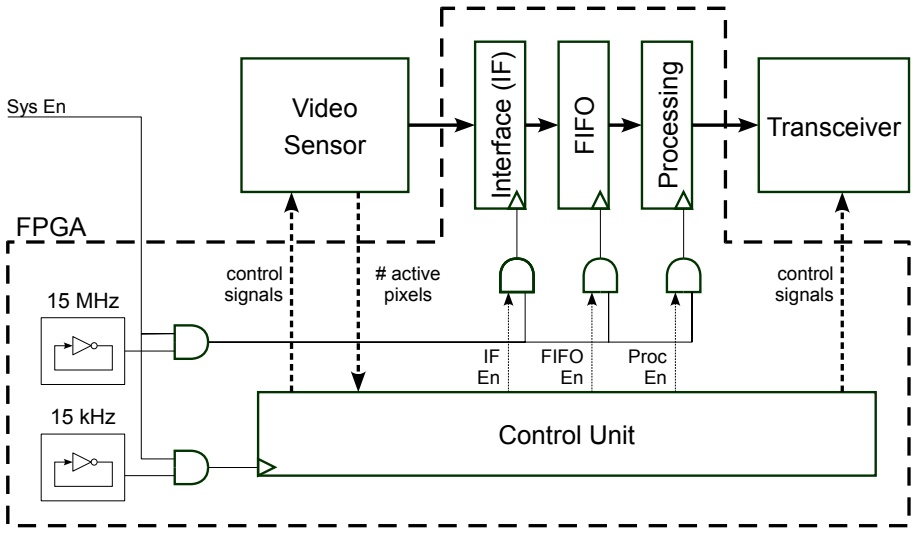


Figure 4.1: Block scheme of the proposed video-node.

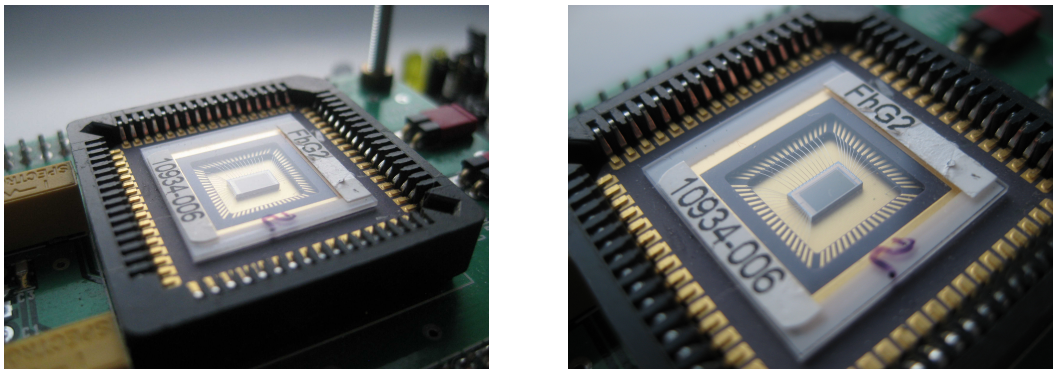
4.1 Hardware

As shown in Fig. 4.1, the proposed video-node is composed of three main hardware elements: the imager, the FPGA, and a transceiver. A detailed analysis of these elements is presented in the following.

4.1.1 Sensing

The imager employed in our system is a prototype called *GrainCam* [10, 69] developed by researchers at the Fondazione Bruno Kessler in Trento (Italy). This is a 128×64 pixels, binary, contrast-based sensor which provides address-based output data asynchronously.

Imaging is achieved by a two-stage process: an analog phase, aiming at the acquisition of the current frame, and a digital phase, in which the output frame is generated. During the acquisition process, each pixel receives a binary value obtained by comparing the incoming irradiance at three locations: the pixel itself, the pixel above and the pixel at its right.



(a) (b)

Figure 4.2: Pictures of the GrainCam vision sensor developed at the Fondazione Bruno Kessler in Trento (Italy).

We refer to this L-shaped structure as the *kernel*. The pixel is said to be *active* if the difference of incoming light between the most and the least irradiated pixels in the kernel exceeds a predefined amount. Thus, at the end of the acquisition process, the active pixels represent the parts of the image characterized by high contrast. In the second part of the process, the sensor computes the pixel-by-pixel (bit-by-bit) difference of the current image with a previously acquired (binary) frame, stored in an internal memory. For instance, we can set a reference frame representing an empty background to achieve a basic background subtraction mechanism, or, by subtracting each frame with the previous one, we can detect the high contrast points in the scene undergoing motion. Fig. 4.3shows some examples of the images that can be produced by this sensor. The images exhibit some isolated active pixels corresponding to points in the scene characterized by a measured contrast close to the threshold. The noise present in the analog section of the system causes the difference of irradiance in the kernel of these pixels to oscillate above and below the threshold, thus generating a blinking effect. Nevertheless, the method allows to effectively detect the foreground elements, as shown in Fig. 4.4.

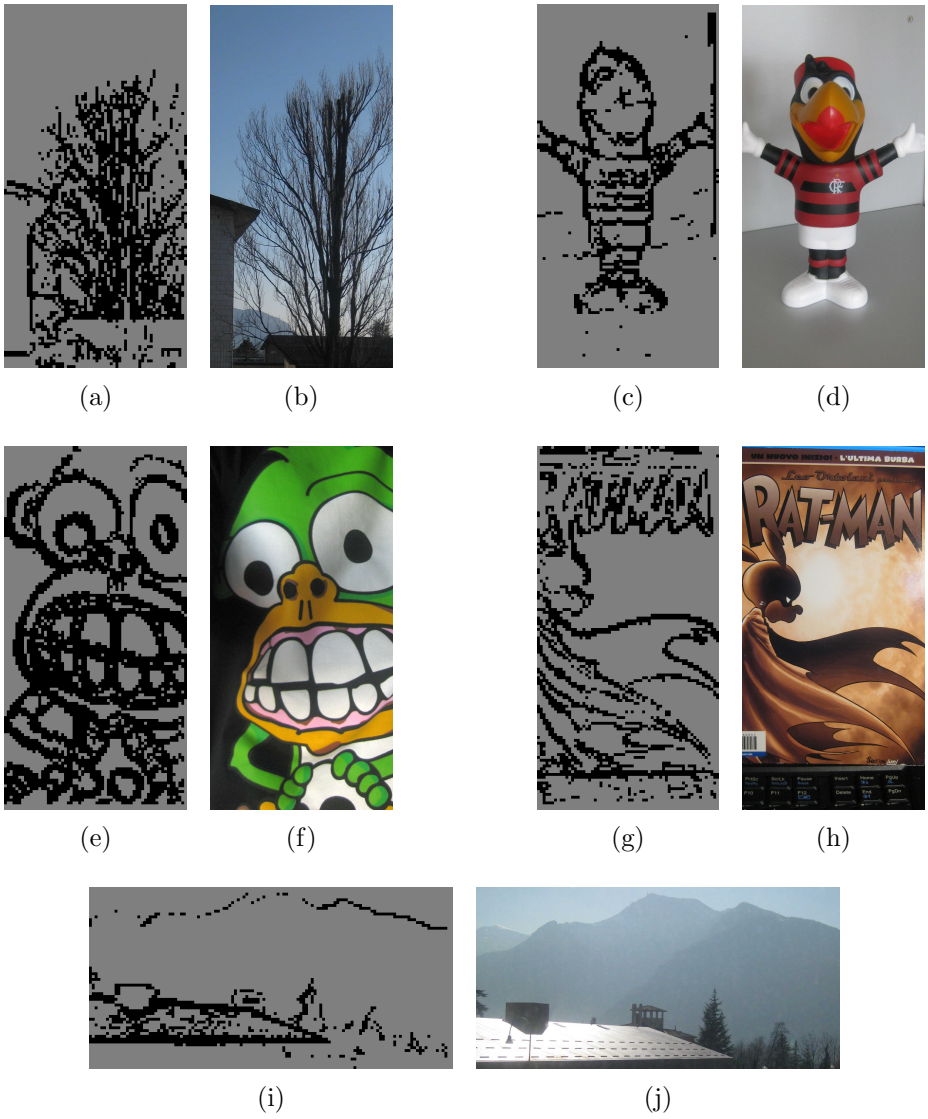
The sensor features the two output modes introduced in Sec. 3.1, and

behaves as follows. When the Active mode is selected, the sensor waits for an external command to start the read-out process, in which the column address and the sign of the non-zero pixels of the difference image are provided at the output pins (the row address is derived from an end-of-row signal). This process is asynchronous, meaning that the difference image is raster-scanned, and an output enabling signal running at 80 MHz is raised every time the data at the chip's output pins represents the address of a non-zero pixel. This process is executed in less than 200 μs. Notice that this kind of output data allows to achieve image compression when the active pixels are not too many. Typically, this is true when the sensor is set to detect motion, as shown in Tab. 4.1. Conversely, when the scene is full of details, this method for compressing data is counter-productive.

In Idle mode, the sensor scans the image and, at the end of the process, it provides at the output pins only the number of non-zero pixels present in the difference image. This number can be used as an indicator of the presence or absence of foreground/moving objects when setting the sensor to operate in background subtraction/motion detection configuration [70].

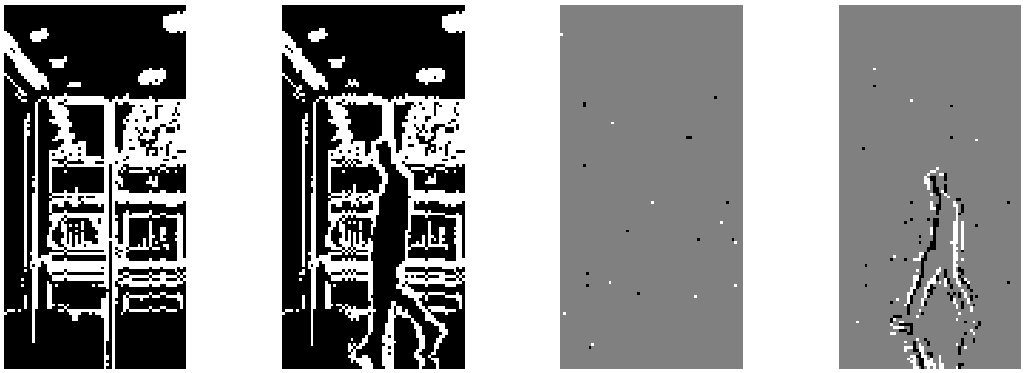
The frame rate can be quite high, since it is only limited by the duration of the acquisition process. Under normal light conditions the integration time required by the sensor can be as low as 5 ms, enabling a frame rate as high as 200 frames/s.

The overall power consumption of the sensor is extremely small. At a frame rate of 50fps, with 25% active pixels, the sensor draws approximately 100 μW when in Active mode. Notice that, since active pixels represent high contrast areas (typically edges), it is unlikely that more than 25% of the pixels are active. If the system is set to Idle mode, the power consumption reduces to 30 μW. This value is over two orders of magnitude less than virtually any other image sensor available on the market.



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j)

Figure 4.3: Images taken by our camera compared with a picture of the same subjects taken with a standard camera: in (a) we can easily identify the tree with its branches and the house at its left; note that the bottom of the tree disappears due to the presence of the roof of another building behind it. This effect is due to the lack of contrast between the tree and the roof, but we can hardly distinguish them in (b) also. Another example is provided in (c), representing the mascot of the Brazilian soccer team Flamengo, shown in (d). The sensor efficiently detects strong, sharp contrasts, as it can be seen in (e) with reference to (f), but it provides very good performance even when the contrast patterns are more complex, as in (i), which represents the detected high contrast points of the landscape in (j).



(a) (b) (c) (d)

Figure 4.4: Images taken by our camera from the same location: (a) and (c) represent a static background, while (b) and (d) represent a person walking by. The result of the acquisition process is given by (a) and (b). The sensor generates (c) and (d) by subtracting the last acquired frame to the current one. Note that in this case, each pixel can take three values, since it is the difference between two binary values.

4.1.2 Control, storage and processing

The core of our video-node is represented by an FPGA-based board. As pointed out in Sec. 2.1.2, FPGA processors have well known advantages in terms of speed and processing power with respect to general purpose microcontrollers, such as those used in standard, low-power wireless network nodes. Furthermore, FPGAs offer several advantages over high performance embedded processors such as the Intel XScale family [37], employed in some motes [35, 9]. For example, they allow for parallel task execution, which can be very advantageous in image processing, where the same mask can be replicated multiple times throughout the image [71], and in system control, where individual components may need to be managed independently. Another advantage of FPGA processors is that they provide easy and fast implementation of several custom components (such as soft memories and CPU soft cores) within the same device. They also can support multiple clock domains, thus enabling separate clocks for different tasks. In particular, flash-based FPGAs have two main advantages with

respect to standard SRAM technology: ultra-low static power consumption and non-volatility. Unfortunately, since flash-based FPGA technology is a few generations behind SRAM technology, it still offers a lower density of logic gates and a lower maximum operating frequency, due to longer signal propagation delays. For our video-node we selected the flash-based Actel IGLOO M1-AGL600, which is characterized by 600k system gates and a static power consumption on the order of tens of micro-watts [72].

4.1.3 Communication

In the current implementation, data is transmitted to a host computer through a RS-232 interface, but the node has been conceived to integrate a wireless module such as the ChipCon CC2420 IEEE 802.15.4-compliant radio, which is described in Sec. 2.1.3. In short, this device communicates at 250 kbps and draws a current of less than 20 mA when transmitting or receiving. Power consumption reduces to less than 0.5 mA when in idle mode and around 20 μA in power-down mode [25].

4.2 Firmware

Within the FPGA we implemented a ring oscillator, a FIFO memory with its interface, the processing unit PU and the controller CU. Two clock domains are implemented, as suggested in Sec. 3.1: one running at a low frequency ($f_{LF} \simeq 15$ kHz), which is fed to the CU, and one running at a higher frequency ($f_{HF} \simeq 15$ MHz), for high speed operations. Both are derived from the ring oscillator's output which is divided through a frequency-divider chain. The CU controls this latter clock through a set of AND gates, as shown in Fig. 4.1. Since the components implemented in the FPGA core need to be enabled in different time intervals, a dedicated gate is provided for each controlled component. The CU also generates the

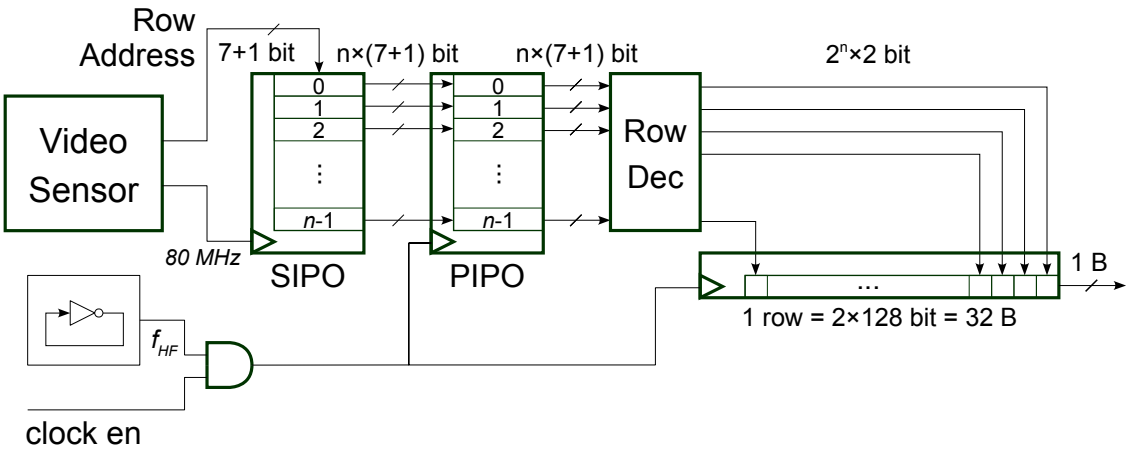


Figure 4.5: Sensor-to-FIFO interface. Data are 8-bit wide and represent the row addresses of the active pixels in the image and their sign. The interface performs a serial to parallel conversion in order to cope with the high rate at which the sensor reads out the data. Addresses are written in the SIPO memory at a maximum rate of 80 MHz. They are then copied into the PIPO memory which is clocked at a frequency $f_{HF} \simeq 15$ MHz. Then, a set of decoders reconstruct the image row. When the end of a row is reached (signaled through the assertion of an End-Of-Row signal), the row is transferred to the FIFO memory.

timing signals for the imager, asks it to perform detection of motion, and selects its output mode, either Active or Idle.

In our prototype the transitions between Active and Idle camera modes are triggered using a threshold n_{pix}^{th} on the number of active pixels in a frame and a waiting time T_W, as described in Sec. 3.2. The value for n_{pix}^{th} has been chosen a little greater than the number of active pixels in a static scene due to the analog noise. Moreover, we chose the value of T_W to be equal to the expected event duration T_D, that is 2 s, which is a reasonable choice since it prevents the system to be switched to Idle mode while an event is still occurring.

When the imager's Active mode is enabled, the data produced by the imager flows at a maximum rate of 80 MHz, which can not be supported

by big soft memories implemented in the FPGA, while adding an external FIFO would increase the node's complexity and power consumption. Therefore, the data goes through the serial-to-parallel interface depicted in Fig. 4.5 that slows down the access to the internal soft memory. The interface also decodes the image converting it from the address-based representation into a bitmap with 2 bpp color depth. This is obtained by means of a 7-bit input Serial Input Parallel Output (SIPO) memory, with a number of elements n such that:

1. $n >= \frac{80\ \text{MHz}}{f_{HF}}$;
2. the corresponding synthesized circuit can run at such speeds.

Of course, the higher n, the lower is the f_{HF} that we can use, but, at the same time, the maximum frequency at which the FPGA allows us to write data on the SIPO memory decreases, since the memory becomes more complex. In our case, things work fine with $n = 8$. While reading out a row of the image, the 7-bit addresses of the active pixels are written in the SIPO memory at a maximum rate of 80 MHz. The pixel sign bits are saved also. Then, the data are copied into a Parallel Input Parallel Output (PIPO) memory, clocked at f_{HF}, in order to allow the SIPO memory to be overwritten without data loss. The first of the two previous constraints on n is required to ensure that no data drops out from the SIPO memory without being first transferred to the PIPO. The addresses and the corresponding sign bits in the PIPO memory undergo the decoding process, through which we reconstruct the current row of the acquired image, which is stored into a 256-bit buffer. In this case, n decoders operate in parallel. It may happen that the same address is decoded multiple times, but this is not an issue since it will always be decoded into the same value. The image row is then safely transferred into the FIFO memory when the End-Of-Row signal is asserted. This process is repeated for every row, and at the end

the FIFO memory will contain the whole image. The soft FIFO is a 16 Kb memory. The data are currently packed in bytes, although the system can be configured for different word sizes. Once the read-out process is over, the PU processes the image to generate the desired output, which is then sent to the transceiver for transmission.

Bandwidth and power issues preclude video streaming and limit image transmission to very few situations. According to the data provided by the manufacturer and the power model present in [1], the transmission of an entire frame (16 Kb) requires about 65 ms while consuming 18 mW. Therefore, the transmission of the whole set of images acquired by one single node at 15fps would fill the available bandwidth, adding a significant contribution to the overall power consumption (about 17 mW).

We can achieve better performance by compressing the images, e.g using a Run-Length Encoder (RLE), which well suits to the employed sensor. The encoding process, implemented in the FPGA, is performed at the same time as the read-out process (thus requiring less than 200 μs), by comparing the addresses of non-zero pixels present at the sensor's output pin. The impact of the RLE in terms of device occupancy is almost negligible, while the reduction of transmitted bytes may be consistent, as shown in Tab. 4.1. When the scene is full of details and the sensor generates a high number of active pixels, it reduces the amount of data by one third with respect to the standard 2 bpp bitmap coding, and halves the size achieved with the address-based representation used by the sensor. If the image is sparse, the RLE cuts the data size to less than 10% of the bitmap image, but in this case the sensor compression method may achieve better rates.

The encoder has been implemented to prove that the node could support image transmission, nevertheless it is not part of the system on which we performed power measurements. This is because the node should be a smart mode, capable of detecting objects, tracking people or perform-

Table 4.1: Compression ratios achieved with the used sensor and a RLE with respect to a bitmap coding scheme with 2 bpp color depth. The images in Fig. 4.4 are considered. In order to better understand the effect of data compression, the percentage of active pixels for each of the images is shown also.

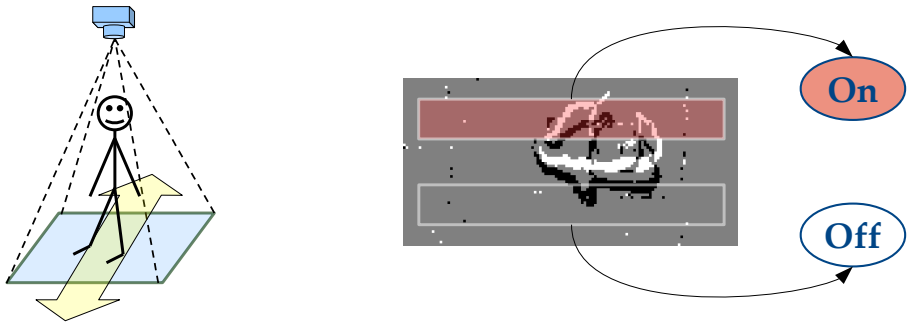
Fig.	4.4(a)	4.4(b)	4.4(c)	4.4(d)
Active pixels	29.1%	28.9%	0.3%	4.9%
Sensor read-out	130.8%	129.9%	1.3%	22.1%
RLE	68.7%	68.3%	5.7%	21.4%

ing any other function required by the application. Therefore, in order to demonstrate the capabilities of the proposed system in a realistic scenario, we configured it to function as a "people counter" [73]. Despite this may not seem a revolutionary application, it is definitely not a trivial task and it requires to implement some intelligence within the node. In the following section we will describe the algorithm and its implementation on the FPGA.

4.2.1 An Integrated People Counter

We developed an integrated system based on the vision sensor described in Sec. 4.1.1, customized as an autonomous counter for persons passing through a "gate", such as a door or a corridor. The algorithm has been developed in cooperation with Roberto Maduchi (University of California, Santa Cruz, USA). The camera node is attached to the ceiling, placed such that its field of view encompasses the width of the corridor or of the door, as depicted in Fig. 4.6(a). Persons are seen entering from the top or bottom of the image and exiting through the opposite end. The task is to count the number of persons transiting in each direction. It should be clear that this is just one possible (and simple) way to use this system; the camera node could be reconfigured for many other monitoring tasks.

Data is processed at full frame rate (30 fps), while using a relatively



(a) (b)

Figure 4.6: Camera node setup (a) and an acquired image representing a person that moves upwards (b). Fig.(b) shows the two implemented VILs. Each VIL can be *on* or *off* according to the number of non-zero pixels that it contains.

simple image analysis procedure. The dynamic characteristics of the application dictate this frame rate. A person walking at normal speed can transit through the field of view of the camera in as little as half a second. Robust estimation of the direction of motion requires analysis of several image frames, hence the need for frame rates in excess of 10 fps. Furthermore, when two persons transit through the field view in close sequence, a high frame rate is necessary to correctly separate the two, otherwise the system may incorrectly count just one passage.

The algorithm for people counting based on the binary data produced by the camera is presented in the following paragraphs.

Image Summarization

We borrow the concept of "Virtual Inductive Loops" (VIL) from Viarani [74], who introduced it for visual traffic monitoring. A VIL mimics the action of an inductive loop, such as those embedded in the road pavement for car counting. In our case, a VIL is simply a particular, typically rectangular region of the image. The system counts the number of asserted pixels in a VIL, reporting a binary 'active' or 'inactive' state for

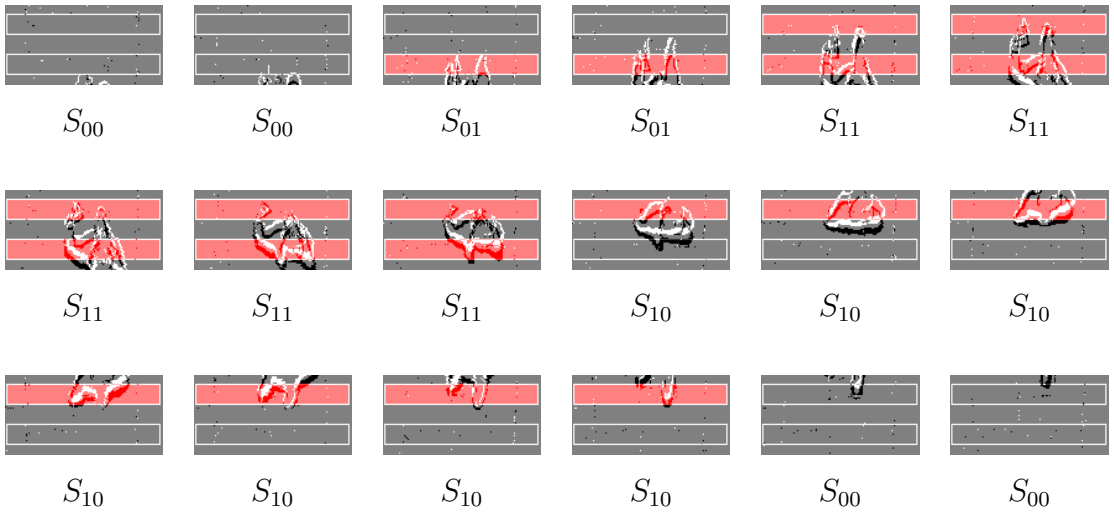


Figure 4.7: An example of a sequence of frames of mode M_{out}, along with the state of each frame. The active VILs are represented in red.

the VIL according to whether the number of asserted pixels is above a fixed threshold. This threshold is chosen empirically based on the noise characteristics of the system. In order to increase robustness, hysteresis is implemented. If the image has N_{VIL} virtual loops, the overall state S is summarized by a binary word with N_{VIL} bits. In our experiments, we used 2 VILs, placed as shown in Fig. 4.6(b) and Fig. 4.7. This is admittedly a crude summarization (a 2-bit *state*) of the binary image content. Yet, this representation is sufficient to accomplish our task with good accuracy. In fact, this summarization procedure has two main advantages. Firstly, the sequence of 2-bit states can be processed very efficiently by the low-power FPGA sensor node architecture. Secondly, counting the number of asserted pixels within a VIL could be achieved *without* the need to buffer the whole binary image data within the FIFO memory. The FPGA could simply count the number of asserted pixels as they are produced by the sensor, checking their address to ensure that it is within either VIL.

In order to simplify our representation further, we group consecutive frames characterized by the same 2-bit state S into *segments* $\sigma = (S, t)$.

t contains the information about the number of frames that make up the segment, i.e., its duration. Thus, the video is represented by the sequence of segments $(\sigma^1, \sigma^2, \ldots, \sigma^N)$, where superscripts represent the index of the segment in the sequence. Accordingly, S^i and t^i represent the state and the duration of a generic segment σ^i. We also define $\Sigma^{i\to j} = (\sigma^i, \sigma^{i+1}, \ldots, \sigma^j)$ and $\mathbf{S}^{i\to j} = (S^i, S^{i+1}, \ldots, S^j)$. When we want to indicate a particular value for a segment's state, we write its 2-bit representation in the subscript (e.g. S^i_{01}). Each segment contains frames with the same state, and two subsequent segments have frames with different states.

We will say that a frame is of *mode* M^m_{in} when it is an image of the m-th person seen in the scene, and this persons is entering the monitored area, i.e., it is transiting from the top to the bottom of the image. Mode M^m_{out} represents the m-th person seen in the scene, who is exiting, i.e., moving from the bottom to the top of the image. In the following, when the index m of the person transiting is irrelevant, we will drop the superscript, in which case mode M_{in} and M_{out} simply represent the fact that a persons is walking in a particular direction. For example, Fig. 4.7 shows a sequence of frames of mode M_{out}[1]. We also introduce mode M_{none} representing sequences with no persons visible in the scene.

An *interval* $\mathbf{I}^{(l)} = \mathbf{I}^{i\to j}$, where the superscript within parentheses denotes an index, is a maximal sequence of segments $\Sigma^{i\to j} = (\sigma^i, \ldots, \sigma^j)$ with the same mode M^m_k, summarized by $\mathbf{I}^{i\to j} = (\Sigma^{i\to j}, M^m_k)$. Note that modes and thus the extent of each interval are not directly observable, and must be inferred from the observable states.

[1]Although the sequence of states in Fig. 4.7 is fairly typical for a mode M_{out}, much more complex sequences can be observed. This is because different parts of one's body may activate the VILs in various orders.

Mode Assignment

The goal of our algorithm is to assign to each segment its correct interval. The algorithm is based on a statistical generative model. We assume that, within an interval, the sequence of segments being generated forms a Markov chain. However, a Markov chain of the first order would not be sufficiently descriptive. A Markov chain of the first order is such that $P(S^n|S^{n-2}, S^{n-1}; M_k^m) = P(S^n|S^{n-1}; M_k^m)$, where $P(S^n|S^{n-2}, S^{n-1}; M_k^m)$ is the probability of the segment state S^n appearing after states S^{n-2} and S^{n-1} in an interval of mode M_k^m. But this is unrealistic in our case, as shown by the following example. The conditional probability $P(S_{01}^3|S_{00}^2; M_{out}^m)$ can be expected to be close to 0.5 (for example, it may correspond to the beginning of the sequence, as a person enters from below and crosses over the second VIL). This is not the case, though, for $P(S_{01}^3|S_{10}^1, S_{00}^2; M_{out}^m)$, which is bound to take a very small value since it is an unlikely sequence during upwards motion. Using a second-order Markov chain removes this ambiguity.

Assume that a generic sequence of segments $\Sigma^{i\to j}$ with states $\mathbf{S}^{i\to j} = (S^i, S^{i+1}, \ldots, S^j)$ is contained in an interval of mode M_k^m (as denoted by the writing $\Sigma^{i\to j} \subset M_k^m$). The likelihood of observing the sequence of states $\mathbf{S}^{i\to j}$ under this assumption is:

$$P(\mathbf{S}^{i\to j}|\Sigma^{i\to j} \subset M_k^m) = P(S^i|M_k^m) \cdot P(S^{i+1}|S^i; M_k^m) \cdot \cdot \prod_{n=i+2}^{j} P(S^n|S^{n-2}, S^{n-1}; M_k^m) \tag{4.1}$$

In addition, we introduce two fictitious (and thus non-observable) segments, $\sigma_I = (S_I, 0)$ and $\sigma_I = (S_F, 0)$, that are assumed to occur at the beginning and at the end (respectively) of each interval. If an observed sequence $\Sigma^{i\to j}$ with $\mathbf{S}^{i\to j} = (S^i, S^{i+1}, \ldots, S^j)$ is assumed to encompass a whole interval (as denoted by $\Sigma^{i\to j} = M_k^m$), then the probability of observ-

ing $\mathbf{S}^{i\to j}$ should be modified as follows:

$$\begin{aligned} P(\mathbf{S}^{i\to j}|\Sigma^{i\to j} = M_k^m) =& P(S_I|M_k^m)\cdot \\ & \cdot P(S^i|S_I; M_k^m)\cdot \\ & \cdot P(S^{i+1}|S_I, S^i; M_k^m)\cdot \\ & \cdot \prod_{n=i+2}^{j} P(S^n|S^{n-2}, S^{n-1}; M_k^m)\cdot \\ & \cdot P(S_F|S^{j-1}, S^j; M_k^m) \end{aligned} \tag{4.2}$$

In practice,

$$P(S^i|S_I, M_k^m) \tag{4.3}$$

and

$$P(S^i, S^{i+1}|S_I; M_k^m) = P(S^{i+1}|S_I, S^i; M_k^m)\cdot P(S^i|S_I; M_k^m) \tag{4.4}$$

represent the probability that state S^i or the sequence (S^i, S^{i+1}) are seen at the beginning of an interval.

$$P(S_F|S^{j-1}, S^j; M_k^m) \tag{4.5}$$

represents the probability that the sequence (S^{j-1}, S^j) is seen at the end of an interval. Note that the marginal probability $P(S_I|M_k^m)$ is equal to 1, since the first state in an interval is always S_I. The first and second order conditional probabilities in (4.2) can be learned from labeled training data sets.

The conditional probabilities in (4.1) and (4.2) are independent of the person index m. However, the segments must all belong to the same mode M_k^m. For example, consider the case with two persons traversing right after each other in the same direction. The first person, with index $m = 1$, is seen in the segments $\Sigma^{1\to N_1}$ with states $\mathbf{S}^{1\to N_1}$ (S^1 through S^{N_1}), while the second person ($m = 2$) is seen in segments $\Sigma^{(N_1+1)\to N_2}$ with states

$\mathbf{S}^{(N_1+1)\to N_2}$ (S^{N_1+1} through S^{N_2}). In this case,

$$\begin{aligned} P(\mathbf{S}^{1\to N_2}|\Sigma^{1\to N_1} = M_k^1, \Sigma^{N_1+1\to N_2} = M_k^2) = \\ = P(\mathbf{S}^{1\to N_1}|\Sigma^{1\to N_1} = M_k^1) \cdot P(\mathbf{S}^{N_1+1\to N_2}|\Sigma^{N_1+1\to N_2} = M_k^2) \\ = P(\mathbf{S}^{1\to N_1}|\Sigma^{1\to N_1} = M_k) \cdot P(\mathbf{S}^{N_1+1\to N_2}|\Sigma^{N_1+1\to N_2} = M_k) \end{aligned} \tag{4.6}$$

where $\mathbf{S}^{1\to N_2}$ is the juxtaposition of the two state sequences, and each of the corresponding sequences of segments is equal to a distinct interval.

In our model we assume that intervals with mode M_{none} contain only one segment. When nobody is walking through the area under surveillance, we expect no state transitions; in particular, we expect the system to be in the S_{00} state, since no motion should be detected and therefore no VIL should be on. In practice, noisy pixels may generate unexpected and uncorrelated state transitions, which are hardly compatible with any of the modes. In order to be able to explain such noisy transitions, we force M_{none} intervals to be only one segment long. In this way a $S_{00} \to S_{01} \to S_{00}$ transition generated by noise would be detected as three distinct intervals of mode M_{none}. In addition, we assume that intervals of mode other than M_{none} have at least two segments Finally, we assume that any given interval cannot have length more than a pre-determined value $T_{\max}$, where $T_{\max}$ is defined in terms of frames, rather than of segments. Basically, $T_{\max}$ is the length (in frames) of the longest observed interval. In our tests, we set $T_{\max} = 60$ frames (2 seconds at 30 fps). These assumptions simplify our algorithm considerably without affecting its accuracy.

Let $\hat{\mathbf{I}}^{i\to j}$ be a candidate interval, consisting of the sequence of segment $\Sigma^{i\to j}$, that has been assigned to mode M_k. We can compute its likelihood as by (4.2)

$$P_{M_k}^{i\to j} = P(\mathbf{S}^{i\to j}|\Sigma^{i\to j} = M_k). \tag{4.7}$$

For every sequence of segments $\Sigma^{1\to N}$, we can build a set of possible intervals $(\hat{\mathbf{I}}^{(1)}, \hat{\mathbf{I}}^{(2)}, \ldots, \hat{\mathbf{I}}^{(L)})$, where the l^{th} interval $\hat{\mathbf{I}}^{(l)} = \hat{\mathbf{I}}^{i(l)\to j(l)}$, is made

up by $\Sigma^{(l)} = \Sigma^{i(l)\to j(l)}$, is assigned to mode $M_{k(l)}$, and has likelihood $P^{(l)}_{M_{k(l)}} = P^{i(l)\to j(l)}_{M_{k(l)}}$. Armed with these definitions, we can state our problem:

Problem: Find the subset of indices $\mathscr{L} \subset \{1, 2, \ldots, L\}$ that maximizes the product of the likelihoods:

$$\prod_{l\in\mathscr{L}} P^{(l)}_{M_{k(l)}} \tag{4.8}$$

under the following constraints:

1. intervals do not overlap, i.e., they do not have any segment in common:

$$\forall l_1, l_2 \in \mathscr{L}, \text{ with } l_1 \neq l_2, \Sigma^{(l_1)} \cap \Sigma^{(l_2)} = \emptyset; \tag{4.9}$$

2. no segment is left uncovered, but all segments belong to an interval:

$$\bigcup_{l\in\mathscr{L}} \Sigma^{(l)} = \Sigma^{1\to N}. \tag{4.10}$$

Differently stated, we find the sequence of people crossing in both directions that maximizes the likelihood of the observed data. Then the number of people who traversed from top to bottom (or viceversa) in a certain period of time is equal to the number of distinct indices $l \in \mathscr{L}$ with $M_{k(l)}$ equal to M_{in} (or M_{out}).

Multigraph representation

The problem can be represented through an acyclic, directed multigraph in which we add a node for each segment σ^i and we draw an edge for every candidate interval $\hat{\mathbf{I}}^{(l)}$ that we take into account (we will then use the terms node and segment and the terms edge and interval interchangeably). More specifically, if the interval $\hat{\mathbf{I}}^{(l)} = \hat{\mathbf{I}}^{i\to j}$ includes the segments $\sigma^i, \sigma^{i+1}, \ldots, \sigma^j$, then the corresponding edge will connect the node σ^i to the node σ^{j+1}.

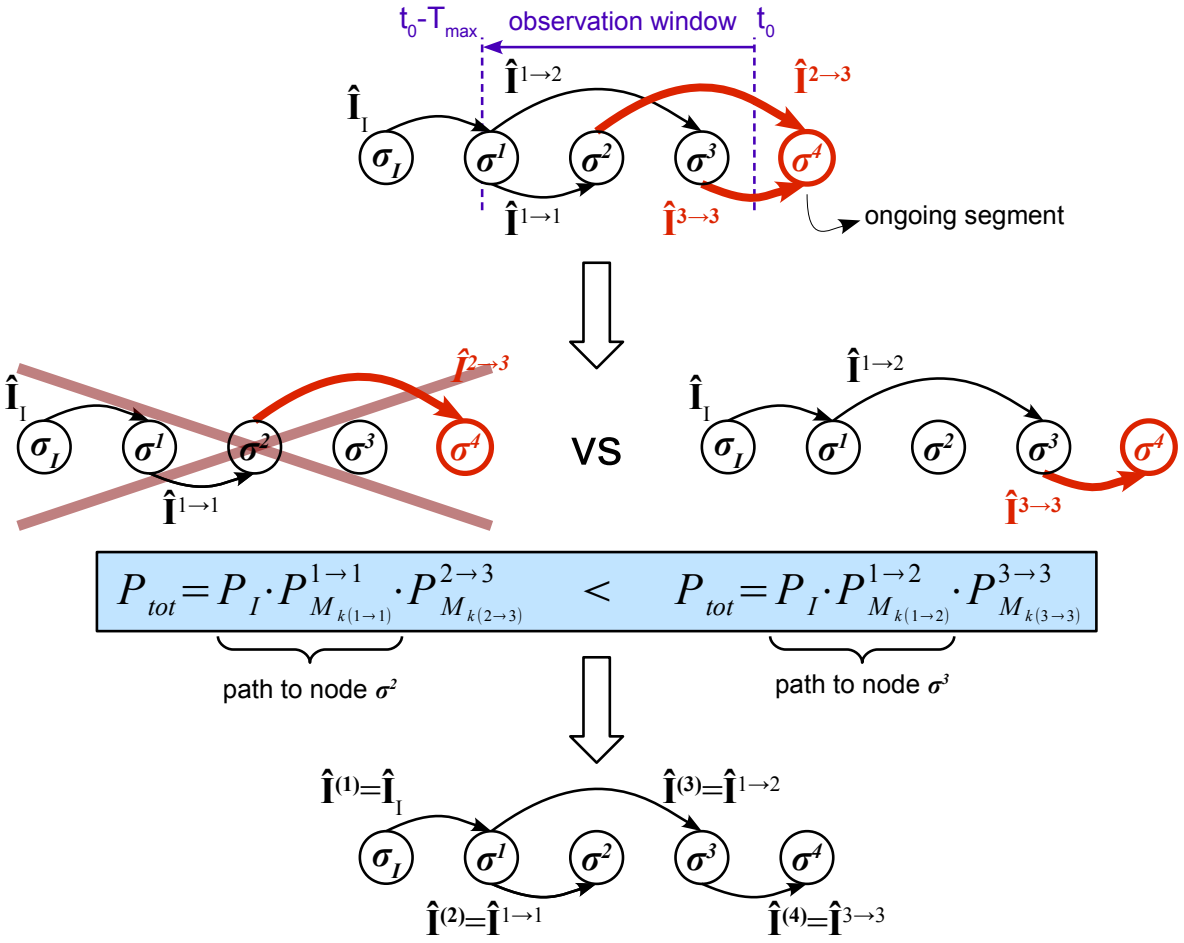


Figure 4.8: The image shows the process of insertion of a new node (σ^4) into the graph. The first step requires to find the new edges (bold arrows) and calculate their cost. Each edge is associated to an interval. For example, the edge connecting σ^2 to σ^4 represents the $\hat{\mathbf{I}}^{2\to3}$ interval, i.e., the interval that includes segments σ^2 to σ^3. No edge can be drawn connecting σ^1 to σ^4 because the $\hat{\mathbf{I}}^{1\to3}$ interval does not fit in the T_{max}-wide observation window (sketched with the dashed vertical lines at the top). Then, all the paths to the current node have to be compared and only the one with maximum cost is kept. Edges to previous nodes are kept too (i.e., $\hat{\mathbf{I}}^{1\to1}$), even if they do not belong to the current path at maximum cost. This is because in the future there may be a highest cost path to another node σ^5 that does not passes through the one that we have just inserted. The resulting graph has one single edge directed inwards for each node σ^n; therefore we can enumerate edges (intervals) using the same index as their destination node.

No cost is associated to the nodes, while the cost of each edge is given by the likelihood $P^{(l)}_{M_{k(l)}} = P(\mathbf{S}^{i(l)\to j(l)}|\Sigma^{i(l)\to j(l)} = M_{k(l)})$ of the interval that it represents. The graph origins on an initial node σ_I meaning that the node just woke up from Idle mode, and ends on a final node σ_F, which represents the instant at which the node switches its mode from Active to Idle, i.e., when a single segment longer than T_{max} occurs. Since the initial state is fictitious, it can not be part of a real interval; we solve this problem with an edge $\hat{\mathbf{I}}_I$ connecting σ_I to σ^1 with cost $P_I = 1$.

Note that the number of edges in the graph is bounded from above by $N_{\text{segm}} \cdot T_{\text{max}}$, where N_{segm} is the total number of segments in the video. This is a very conservative upper bound, since it is highly unlikely that all segments are made up by a single frame. In practice, within an interval of T_{max} frames, there are about 5 segments on average.

Thus, our problem becomes one of finding the maximum cost path in the multigraph from the first to the last node (segment), where the cost of a path is the product of the costs of the edges in the path. Since all costs are positive and less than or equal to one, this problem is equivalent to one of finding a minimum cost path where the path cost is the sum of the (non-negative) negative logarithms of $P^{(l)}_{M_{k(l)}}$. Note that two nodes have multiple edges linking them, one for M_{in} and one for M_{out}, but only the edge with the maximum cost needs to be kept. Dijkstra's algorithm can then be used to find the maximum cost path in the resulting graph, with a complexity of $\mathcal{O}(N_{\text{segm}})$ (since the number of edges grows linearly with the number of nodes). In the following, we will describe the procedure, with reference to Fig. 4.8.

At every state transition, a new segment σ^{n+1} is generated, thus we insert a new node into the graph. Then, we create edges to it, i.e.,, according to our representation, we find all the possible intervals that end with the segment σ^n (we cannot use σ^{n+1} within an interval since it is still ongoing

and we do not know its duration t^{n+1}, but this is fine with the way we build an edge) and calculate their cost, i.e., the likelihood of the associated interval. The graph is such that if there exist an edge from node Σ^{n-p} to Σ^n, then all the edges from node Σ^{n-r} with $r < p$ exist too, since they include less segments. Viceversa, if we cannot build an edge from node Σ^{n-p} to Σ^n, then we can not build any edge from a node Σ^{n-r} with $r > p$ either. Once we have found the edges, we need to compare all the paths (there is one path for each edge) that bring to node σ^{n+1} and keep only the one that maximizes the cost. Since we keep track, for each node σ^{n-p}, of the highest cost path $P^{path\ to\ \sigma^{n-p}}$ to it, this task simply involves multiplying the cost of each newly generated edge $\hat{\mathbf{I}}^{(n-p)\to n}$ by the total cost of the path up to σ^{n-p} (which is the source node of that edge). At the end, there will be only one edge $\hat{\mathbf{I}}^{(l)}$ directed inward for each node σ^n; therefore we can re-enumerate the intervals by choosing $l = n$. All we need to remember about each edge are: its source node $\sigma^{n-p(n)}$, the total cost $P^{path\ to\ \sigma^n}$ of the path it belongs to, and the mode $M_k(n)$ associated to the interval that it represents. When the system state does not change for a period longer than T_{max}, the node switches to Idle mode. This is translated into a σ_F node which is added to the graph. The highest cost path leading from σ_I to σ_F then represents the set of intervals that are more likely to have generated the observed sequence of states.

Experimental Evaluation

Two videos (Video 1 and Video 2) were acquired with the binary camera. Each video was hand-labeled, resulting in a list $\mathscr{I} = \left\{\mathbf{I}^l\right\}_{l=1}^{L}$ of "ground truth" intervals (the l^{th} interval is labeled with mode $M_{k(l)}$). In some instances, more than one person was seen at the same time in the camera's field of view. These sequences are labeled with a mode of type M_{other}. Note that our algorithm can only identify sequences of mode M_{none}, M_{in} or M_{out},

so a situation with M_{others} will produce an error. Video 1 contained 186 intervals, 23 of which of type M_{others}. Video 2 contained 193 intervals, 37 of which of type M_{others}. The two videos were taken in different locations with different light conditions. In the case of Video 1, persons transiting in the scene casted shadows that were well visible in the binary images. No shadows were noticeable in Video 2.

Method of Benchmarking

Our algorithm produces a list of intervals $\hat{\mathscr{I}} = \left\{\hat{\mathbf{I}}^n\right\}_{n=1}^{N}$ with the n^{th} interval of mode $\hat{M}_{k(n)}$. In order to assess the quality of classification, one needs a way to compare the list of hand-labeled and automatically produced intervals. The approach we use is to find the set of unique ordered assignments between elements of $\mathscr{I}$ and of $\hat{\mathscr{I}}$ that maximizes a certain criterion. By "unique" we mean that one interval of $\mathscr{I}$ can be assigned to at most one interval of $\hat{\mathscr{I}}$, and vice-versa. By "ordered" we mean that if $\mathbf{I}^{l_1}$ is assigned to $\hat{\mathbf{I}}^{n_1}$ and $\mathbf{I}^{l_2}$ is assigned to $\hat{\mathbf{I}}^{n_2}$ with $l_2 > l_1$, then $n_2 > n_1$. The criterion to be maximized is the sum of the overlap (in frames) of the sequences in each assignment. More precisely, let the assignments be represented by the set of J pairs (l_j, n_j), and let O_j represent the overlap (in frames) between $\mathbf{I}^{l_j}$ and $\hat{\mathbf{I}}^{n_j}$. We seek the unique ordered assignments that maximize $\sum_{j=1}^{J} O_j$. This task is equivalent to the problem of *global sequence alignment*, a well-studied topic in bioinformatics. We use the Needleman-Wunsch algorithm [75], with scores equal to O_j and zero gap penalty. This algorithm solves global sequence alignment using dynamic programming. Once the sequences are aligned, we compute the rate of "correct detection" (matches $[\mathbf{I}^l, \hat{\mathbf{I}}^n]$ with $M_{k(l)} = \hat{M}_{k(n)}$), "misses" (elements of $\mathscr{I}$ that do not have a match in $\hat{\mathscr{I}}$) and "false alarms" (sequences of $\hat{\mathscr{I}}$ that do not have a match in $\mathscr{I}$).

Classification Performances

Experimental tests were conducted with one of the two videos used as training set (to learn the state conditional probabilities) and the other one for testing the algorithm. As mentioned earlier, the videos contained intervals with more than one person visible in the scene (M_{others}). These intervals cannot be correctly identified by our algorithm, which is only trained on modes M_{in} and M_{out}. We presents two sets of results. The first set is computed after removing all intervals of type M_{others} from the videos; the second set includes intervals of type M_{others}. The results of the first test are shown in Tab. 4.2 in terms of Error rate, Missed detection rate, and False alarm rate. It is seen that, although the error rate is negligible, some missed detections and false alarms are reported. Upon visual analysis of the data, it was ascertained that occurrences of missed detection were mostly due to poor lighting condition. The rate of false alarms is negligible when the algorithm is trained on Video 1 and tested on Video 2, but fairly high (14%) when the training and test sets are reversed. The reason for this is that, as observed earlier, Video 1 contains several visible moving shadows. These shadows are sometimes incorrectly interpreted as additional people in the scene by the algorithm when trained on Video 2, which did not have moving shadows.

As expected, if intervals with mode M_{others} are considered (Tab. 4.3), the error rate grows dramatically. The algorithm, when presented with an interval of mode M_{others}, typically assigns one or more intervals of type M_{in} and M_{out} to it, resulting in errors and possibly false alarms. This also explains why the rate of false alarms increases with respect to the previous test. Note that introducing intervals of type M_{others} also increases the missed detection rate when the system is trained on Video 1.

Table 4.2: Classification performances (M_{others} intervals removed)

Training set	Video 1	Video 2
Test set	Video 2	Video 1
Error rate	0%	0%
Missed detection rate	3%	5%
False alarm rate	0%	14%

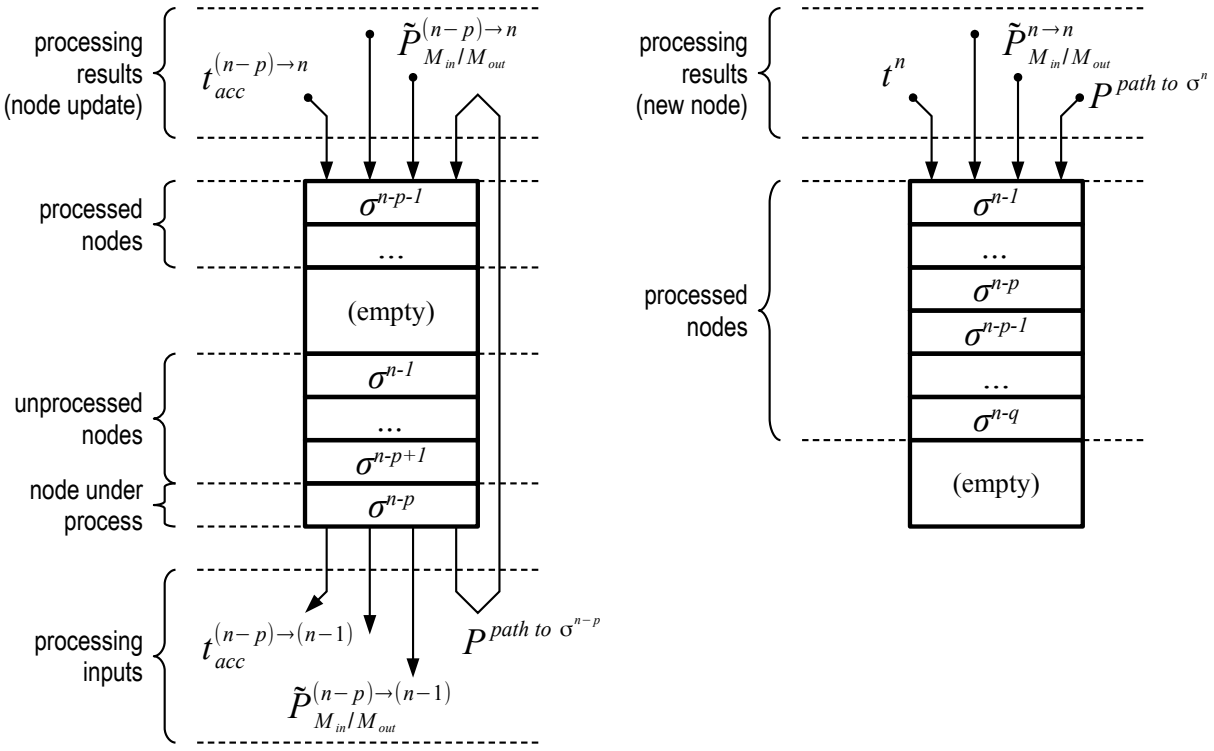
Table 4.3: Classification performances (M_{others} intervals included)

Training set	Video 1	Video 2
Test set	Video 2	Video 1
Error rate	15%	14%
Missed detection rate	8%	5%
False alarm rate	15%	7%

FPGA implementation

In hardware, the previously described algorithm is carried out with the following architecture [76]:

- an *VIL-based image processing unit*, that extracts the current system state from the acquired image;
- two *look up tables (LUTs)*, containing the transition probabilities $\check{P}^{\rightarrow n}_{M_k}$ (indicating a generic transition to state S^n); one LUT is for the first order transition probability maps, containing the $P(S^1|S_I; M_k)$ probabilities, and the other is for the second order transition probability maps, from which we can get the values for $P(S^n|S^{n-2}, S^{n-1}; M_k)$ and $P(S_F|S^{n-1}, S^n; M_k)$; since M_{none} intervals are all one segment long, the LUT provides directly the total likelihood $P^{(l)}_{M_{none}}$;
- a *three cell memory*, that keeps track of the states of the last segments for LUT addressing; this is a three cell memory because we are dealing with a Markov process of the second order, therefore we need to remember S^n, S^{n-1}, and S^{n-2};



(a) Node memory - node update

(b) Node memory - new node insertion

Figure 4.9: *Node FIFO memory.* Fig. (a) shows its status in the middle of the processing phase, while computing the path to node σ^n through a generic node σ^{n-p}. The pieces of information about σ^{n-p} are read from the memory and provided to the multipliers to calculate the cost of intervals and paths. Once multiplications are performed, the data about σ^{n-p} are written back into the memory, with the values updated. Fig. (b) depicts the last passage of the processing phase, when all previous nodes have been processed and the newly generated one is written in the memory.

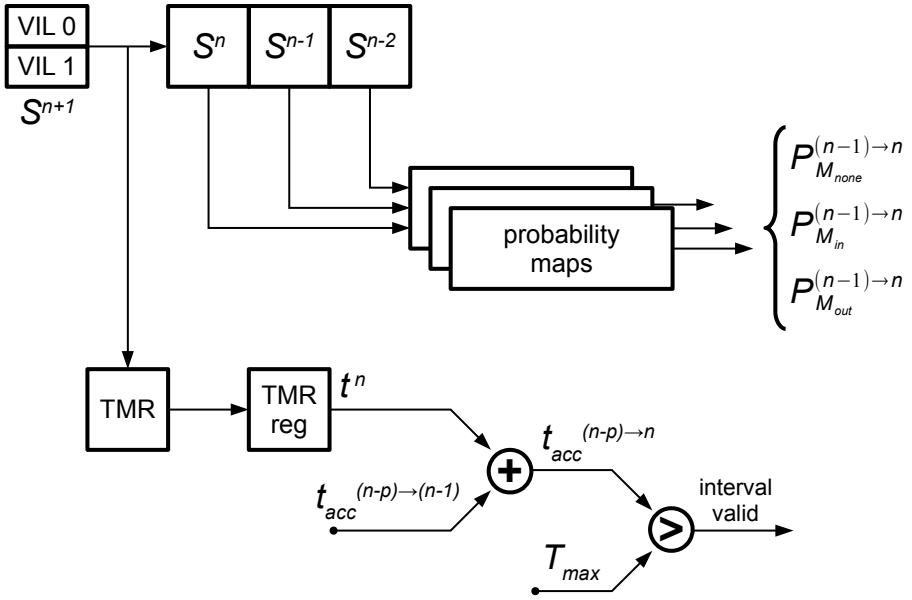


Figure 4.10: Segment generation. This figure depicts the architecture of the *VILs*, the *three cell memory* and the *LUTs*, the *timer* with the *adder* and the T_{max}*-comparator*. This portion of the processing unit monitors the system state through the VILs, and when a state transition to S^{n+1} occurs, the timer is sampled before being reset and the previous state S^n is written into the three cell memory. This memory behaves as a shift register, so that we keep track of the three previous states (S^{n-2}, S^{n-1}, S^n). These pieces of information are used to address the LUTs to obtain the transition probabilities of the first and of the second order, including the transition from the initial state S_I and to the final state S_F. Note that, since M_{none} intervals are only one-segment long (i.e., σ^n), the LUTs provide directly the likelihood for $\hat{\mathbf{I}}^{(n)}_{M_{none}}$. While processing the new segment, at iteration p, the adder sums the timer sampled value with the cumulative duration of previous segments $\sigma^{n-p}, \ldots, \sigma^{n-1}$, which is stored in the node memory, to check if the overall sequence $\sigma^{n-p}, \ldots, \sigma^{n-1}, \sigma^n$ fits within the T_{max}-wide observation window. If this is the case, then the process goes on; otherwise the σ^{n-p} node is dropped from the node memory and the process starts over with the following element σ^{n-p+1}

.

- a *timer*, that measures the duration t^n of segments, and an associated *adder* to calculate the total duration of intervals $t^{(n-p)\to n}$;
- a *comparator*, that determines if a group of contiguous segments is longer than T_{max};
- a *node First In, First Out (FIFO) memory*, that keeps track of the most recent nodes in the graph, i.e., the ones that lay within the T_{max}-wide observation window;
- two *edge multipliers* that calculate the likelihood of M_{in} and M_{out} intervals (no multiplier is required for M_{none} intervals, since they all are one segment long by definition);
- a second *comparator*, that, given a sequence of contiguous segments, determines if they more likely represent either an M_{in} interval or an M_{out} one;
- a *path multiplier* that calculates the total cost of paths;
- a *comparator* that extract the path with the maximum cost;
- an *edge FIFO memory*, that stores the results of the process;
- *registers* and *multiplexers*, to store results and route signals;
- a *control unit*, split in several sub-units, that manages the whole process.

Fig. 4.9, Fig. 4.10, Fig. 4.11 and Fig. 4.12 represent the block diagram for the architecture.

Every time a state transition occurs, we insert the node σ^{n+1} in the graph and find the maximum cost path to it. Node insertion is carried out by saving the VIL state in the three cell memory and the timer value into a dedicated register before resetting them for the next acquisition. Then,

in order to build the first edge, we extract the last node contained in the node's memory (let's call it node σ^{n-q}). The node data is given by:

- the cumulative period of time $t_{acc}^{(n-q)\to(n-1)}$ including nodes from σ^{n-q} up to node σ^{n-1};
- the partial likelihoods $\tilde{P}_{M_k}^{(n-q)\to(n-1)} = P(\mathbf{S}^{i\to j}|\Sigma^{i\to j} \subset M_k^m)$ of the M_k intervals that origin from the node itself and contain nodes up to σ^{n-1};
- the cost of the maximum cost path from σ_I to the node σ^{n-q} itself.

The partial likelihoods are stored in the node memory in order to avoid to perform the same calculations over and over again at every node insertion. In fact, an interval that begins with segment σ^i and ends with σ^{i+j} shares the same probabilities with another interval that origins on the same segment but ends on σ^{i+j+1}. The way the node memory is organized is depicted in Fig. 4.9(a).

The adder sums the value provided by the timer to the cumulative period and the result $t_{acc}^{(n-q)\to n}$ is then compared with T_{max}. If it is greater, then the σ^{n-q} node is discarded from the node memory and the following node (the σ^{n-q+1}) is read. Otherwise, the sequence of segments $\sigma^{n-q}, \ldots, \sigma^n$ represents a valid interval and we have to compute its likelihood $P_{M_k}^{(n-q)\to n}$ for each M_k. Therefore, we fetch the transition probabilities $\check{P}_{M_k}^{\to n}$ stored in the LUTs using the content of the three cell memory for addressing. This is shown in Fig. 4.10.

Then, we multiply the so obtained transition probabilities by the partial cost $\tilde{P}_{M_k}^{(n-q)\to(n-1)}$ to obtain the partial cost $\tilde{P}_{M_k}^{(n-q)\to n}$ of the intervals that include the segments from σ^{n-q} to σ^n. In the event that one further segment is generated, we write back to the node memory the data about the σ^{n-q} node, with the updated values for t_{acc} and the partial likelihoods. To obtain the total cost, $P_{M_k}^{(n-q)\to n}$, we need to add the final state S_F to the

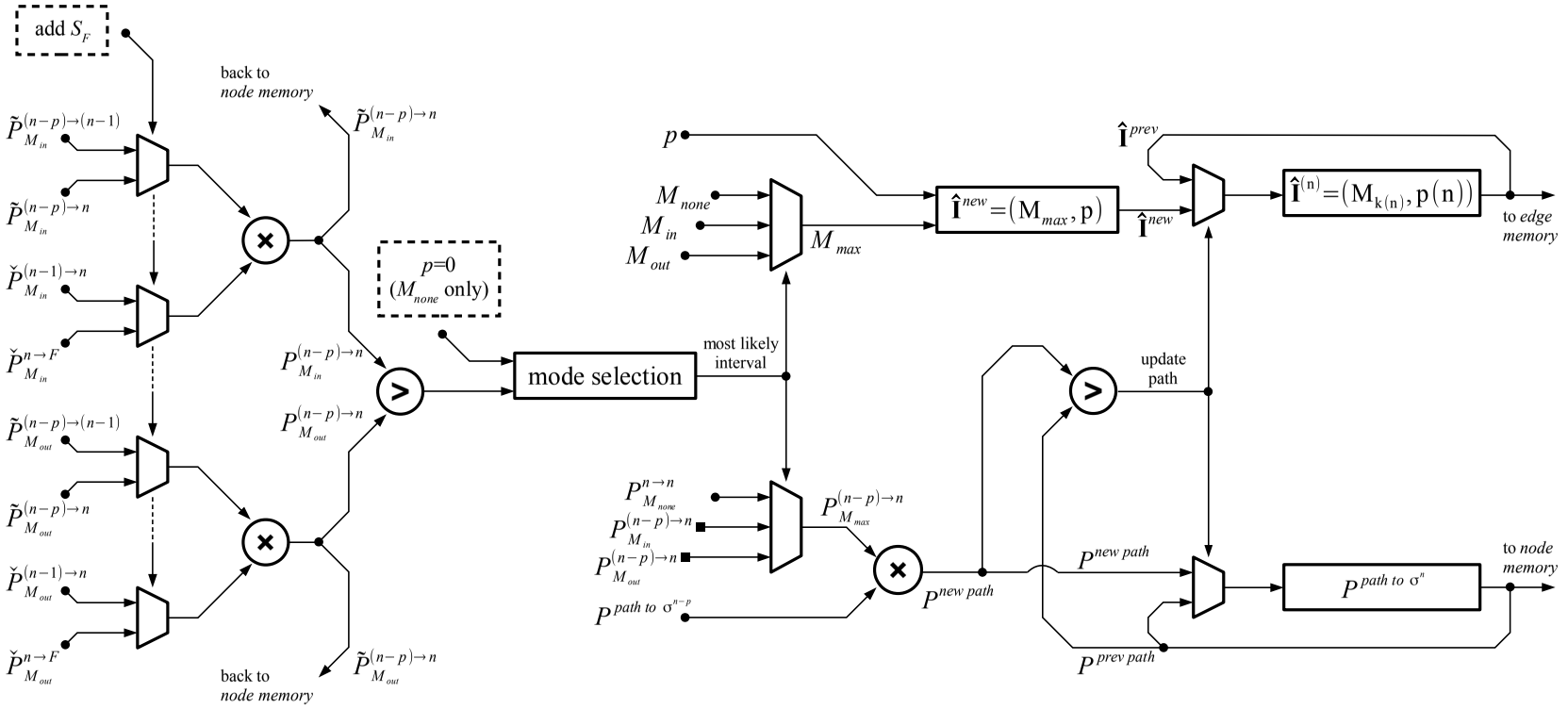


Figure 4.11: Architecture of the processing elements that calculate the maximum cost path to a new node σ^{n+1}. The entire process requires: (1) to calculate the most likely mode M_k for the interval constituted by the segments $\sigma^{n-p},\ldots,\sigma^n$; the partial likelihood $\tilde{P}^{(n-p)\to(n-1)}_{M_k}$ is multiplied by the probability associated to the last state transition $\check{P}^{\to n}_{M_k}$; the resulting $\tilde{P}^{(n-p)\to(n)}_{M_k}$ is stored back into the edge memory, and it is also multiplied by the probability that S^n is seen at the end of the interval to get its likelihood $P^{(n-p)\to n}_{M_k}$; (2) to multiply the interval likelihood by the cost of the path from σ_I to σ^{n-p} to achieve the cost $P^{new\ path}$ of the entire path to σ^n through σ^{n-p}; (3) to compare the so obtained edge with the one with maximum cost $P^{prev\ path}$ resulting from the nodes processed so far and, in case, overwrite it along with the information associated to the edge. Signals included within dash boxes come from the control unit.

M_{in} and M_{out} intervals. This operation involves another data fetch from the LUT and another multiplication. In the next step, the two intervals are compared and only the most likely one $hatI_{M_{max}}^{(n-q)\rightarrow n}$ is kept into account. The likelihood of the so achieved interval $P_{M_{max}}^{(n-q)\rightarrow n}$ is multiplied with the cost of the path to σ^{n-q} by the interval multiplier to achieve the cost of the path to σ^{n+1}. The result is temporarily stored in the last comparator's output register, along with the corresponding mode M_{max} and the index q. Fig. 4.11 describes the hardware architecture that carries out these operations.

Then, the process starts over with a new node, the σ^{n-q+1}, thus generating a different path to node σ^n. Its cost $P^{new\ path}$ is compared with the previous one $P^{prev\ path}$ and of the two, only the one with higher cost is kept. Iteratively, all the nodes σ^{n-p}, with $0 < p \leq q$, that are contained within the node memory are processed. One further interval is created, consisting in the only σ^n segment. The last node involves slightly different calculations, since the only one-segment-long interval allowed represents an event of mode M_{none}. Nevertheless, the partial likelihoods are calculated anyway, since they need to be inserted in the node's memory as part of the new element.

At the end of the process, the path comparator provides the data about the maximum cost path. Its cost is saved into the node memory of Fig. 4.9(b) along with the partial likelihoods and the timer's value as a new element, while the mode and the source of the last interval in the path (i.e., the data about the last edge) are saved into the edge memory present in Fig. 4.12. Then the system waits for another segment.

When the timer exceeds T_{max} the process is interrupted and the control unit can extract the intervals that are more likely to have happened by reading the data from the edge memory. Each element will contain the mode associated to the edge and the pointer to the previous valid element

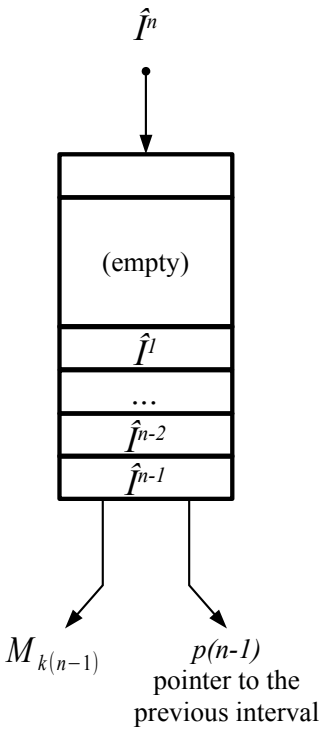


Figure 4.12: *Edge FIFO memory.* After all the nodes contained the node memory have been processed, the edge $\hat{\mathbf{I}}^{(n)}$ maximizing the cost path to σ^n is written in the edge memory (while the pieces of information associated to node σ^n (i.e., t^n, $\tilde{P}_{M_k}^{n\to n}$ for $M_k = M_{in}, M_{out}$, and $P^{path\ to\ \sigma^n}$) are written into the node memory as a new element). The $\hat{\mathbf{I}}^{(n)}$ is described by its mode $M_k(n)$ and the pointer to its source node $p(n)$. When the final segment σ_F is inserted into the graph and the maximum cost path to is built, the edge memory is read to extract the occurrences of M_{in} and M_{out} that are more likely to have happened.

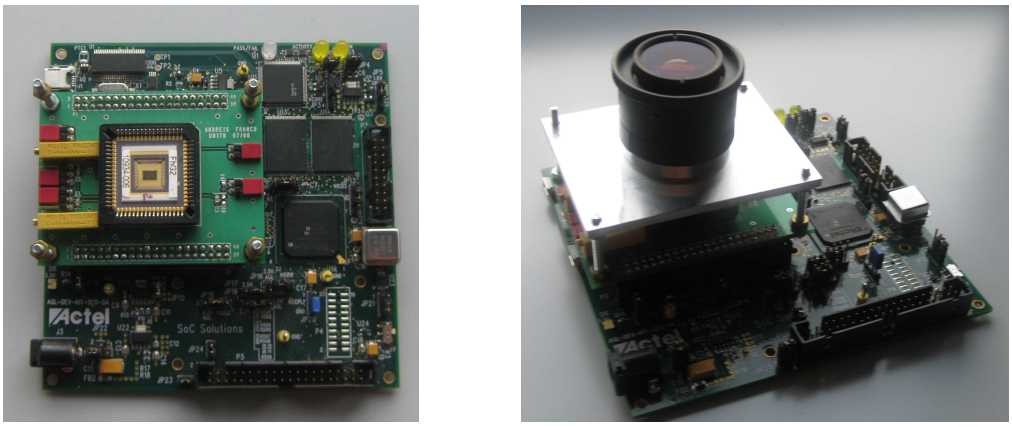
in the memory. This is like going back along the maximum cost path jumping from the first segment of an interval to the one of the previous interval.

This architecture, implemented on the Actel IGLOO FPGA, has been tested with a clock running at 10 MHz. At such an operating frequency, the system can safely acquire images at 30 fps. The whole implementation occupies 5101 out of 13824 VersaTiles (36%). VersaTiles are Actel Igloo's basic elements which can be customized to generate the required function. The implementation also uses 48 out of 235 I/O lines (20.43%).

4.3 Power Measurements and Lifetime Estimation

In the following, the power consumed by the system when executing the people counting algorithm described in Sec. 4.2.1 is analyzed. We implemented the prototype on a commercial development board mounting several components in addition to the Actel M1-AGL600 FPGA: a 1 MB SRAM, a 16 MB Flash memory, a crystal oscillator, a programmer, a USB to RS-232 converter chip, expansion connectors, LEDs and switches. We connected the camera to the board through one of the available connectors. The node communicates to a host computer through a USB connection that implements the RS-232 interface.

On the host PC's side, we developed a program in Python using the WxWidgets library to create the graphical user interface (GUI). A snapshot of the GUI is presented in Fig. 4.14. From the GUI we can set parameters such as integration time and frame rate, change the threshold for the number of active pixels that switches the system from active to idle mode and viceversa. Commands for starting and stopping the system can be sent to the FPGA by means of simple buttons, along with controls for selecting the frame to store in the sensor's internal memory (let it be an



(a) (b)

Figure 4.13: Pictures of the developed prototype node. The board integrates an Actel Igloo M1-AGL-600 FPGA, a 1 MB SRAM, a 16 MB Flash memory, a crystal oscillator, a programmer, a USB to RS-232 converter chip, expansion connectors, LEDs and switches. The sensor is mounted on a daughter board and communicates with the FPGA through an expansion connector. The board allows to measure the current drawn by the FPGA core, its I/O banks and the imager separately.

empty frame if we want to see the acquired image, or a user selected frame representing the background, or every acquired frame to detect moving objects). The GUI allows to see the sensor output image and the result of the people counting algorithm.

The development board allows for measurement of the current flowing through the FPGA core, its I/O banks and the camera separately. We omitted the contribution due to the wireless module, partly because the energy required for transmission and reception varies significantly according to the nodes positioning and the amount of data to transmit.

Measurements were carried out by forcing the system to run in one of the configurations described in Sec. 3.3. When the node switches from Idle to Active mode the system state needs to be continuously monitored. In our implementation, the execution of this task is performed during the readout process, thus it would not require a soft memory. Nevertheless, in order to

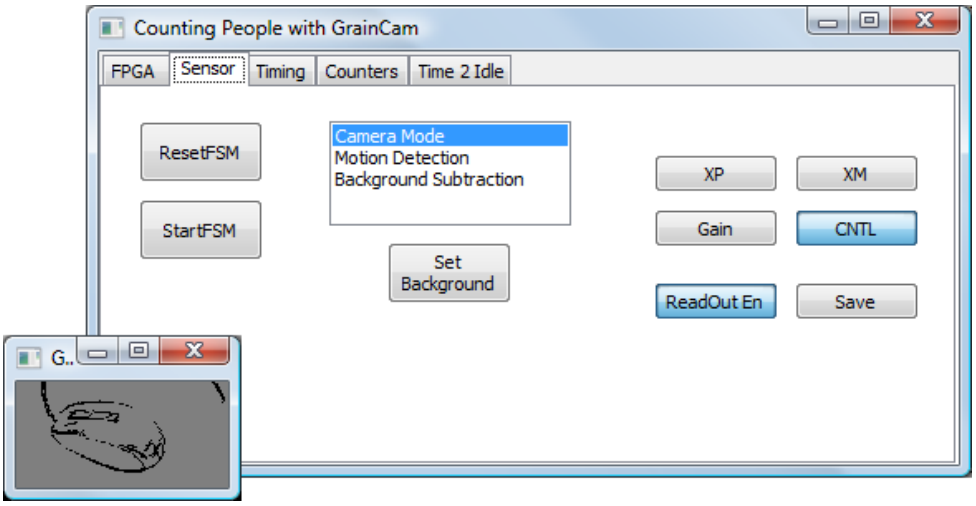


Figure 4.14: Graphical User Interface running at the host PC's side. This is a Python program written with the WxWidgets library. It allows to control the prototype node and see the output coming from the sensor. It also allows to see the result of the people counting algorithm. The acquired image represents a computer mouse.

be able to monitor the behavior of the node, we programmed the FPGA to transmit the whole acquired image to a host pc at every frame, thus the soft memory has been finally implemented. We set the transmission rate through the RS-232 interface to 256 Kbps. At this speed, the system could not support acquisition rates higher than 10 fps, therefore the frame rate had to be reduced. We do not expect a significant power consumption increment at 30 fps, i.e., the frame rate which the algorithm has been designed for. This is because the control unit in the FPGA enables the HF clock for all the time needed for image transmission over the RS-232 interface, which takes much longer than the execution of the algorithm. In this way, large portions of the FPGA are active for most of the time, thus reducing the benefits of employing a flash-based FPGA.

We employed two Agilent 34411A digital multimeters to perform power measurements. We set one of them to operate as an ammeter with range 10 mA and an integration time equal to 100 power line cycles, i.e., 2 s. With these settings, the instrument measures the voltage drop on a shunt

resistance of 2 Ω. We measured the voltage drop on the device with the other multimeter, operating on a range of 10 V with an integration time of 100 power line cycles, and characterized by an input resistance greater than 10 GΩ. With this configuration the instruments' normal mode noise rejection is maximized, and their loading effects are negligible. Results are shown in Tab. 4.4.

When the sensor is set to operate in Idle mode, the FPGA and the sensor consume as little as 2.50 mW. In this case, the FPGA I/O banks represent the major contributor to the overall power consumption (45%). The measured power consumed by the sensor is much higher than the nominal one declared in [10]; this is due to the presence of trimmers and other components that are present on the board for debugging purposes.

In active mode, when the system state does not change and the only feature extraction phase is performed, things change considerably, with the node consuming 4.08 mW. The sensor almost doubles its contribution, since it needs to drive the FPGA input pins to read out the image. The FPGA core draws more than twice the current than in Idle mode, due to the activation of the high frequency clock while receiving the image, performing the extraction of the features from it and then transmitting it through the RS-232 interface. We believe that major contribution is given by the FIFO memory, which is activated for most of the frame period, i.e., both while receiving the image from the sensor and while processing/transmitting it. Nevertheless, the impact of such increase in current drawn is reduced by the fact that the FPGA core is powered at 1.2 V, while the rest of the node is powered at 3.3 V. Little change occurs for the FPGA I/O banks (just a 15% increase in current draw), which is somehow unexpected since image transmission requires to drive the USB to RS-232 converter chip serial input pins. A possible explanation lays in the fact that the image is binary and most of the pixels are equal to '0', therefore the FPGA serial output

Table 4.4: Supply voltage and current drain for our prototype node. The values related to the video-sensor also comprise the consumption of other elements present on the same board, such as trimmers and other resistors required for debugging.

	$V \pm \Delta V$ [V]	$I \pm \Delta I$ [mA]
Video-sensor	3.28±0.02	0.26±0.01
FPGA core	1.22±0.01	0.44±0.02
FPGA I/O banks	3.28±0.02	0.34±0.02
Overall power P_{I}	2.50 ± 0.14 mW	

(a) Idle Mode

	$V \pm \Delta V$ [V]	$I \pm \Delta I$ [mA]
Video-sensor	3.28±0.02	0.45±0.02
FPGA core	1.22±0.01	1.09±0.06
FPGA I/O banks	3.28±0.02	0.39±0.02
Overall power $P_{\mathrm{A,FE}}$	4.08 ± 0.23 mW	

(b) Active Mode - Feature extraction only

	$V \pm \Delta V$ [V]	$I \pm \Delta I$ [mA]
Video-sensor	3.28±0.02	0.45±0.02
FPGA core	1.22±0.01	1.15±0.06
FPGA I/O banks	3.28±0.02	0.41±0.02
Overall power $P_{\mathrm{A,DM}}$	4.22 ± 0.23 mW	

(c) Feature Extraction and Decision Making

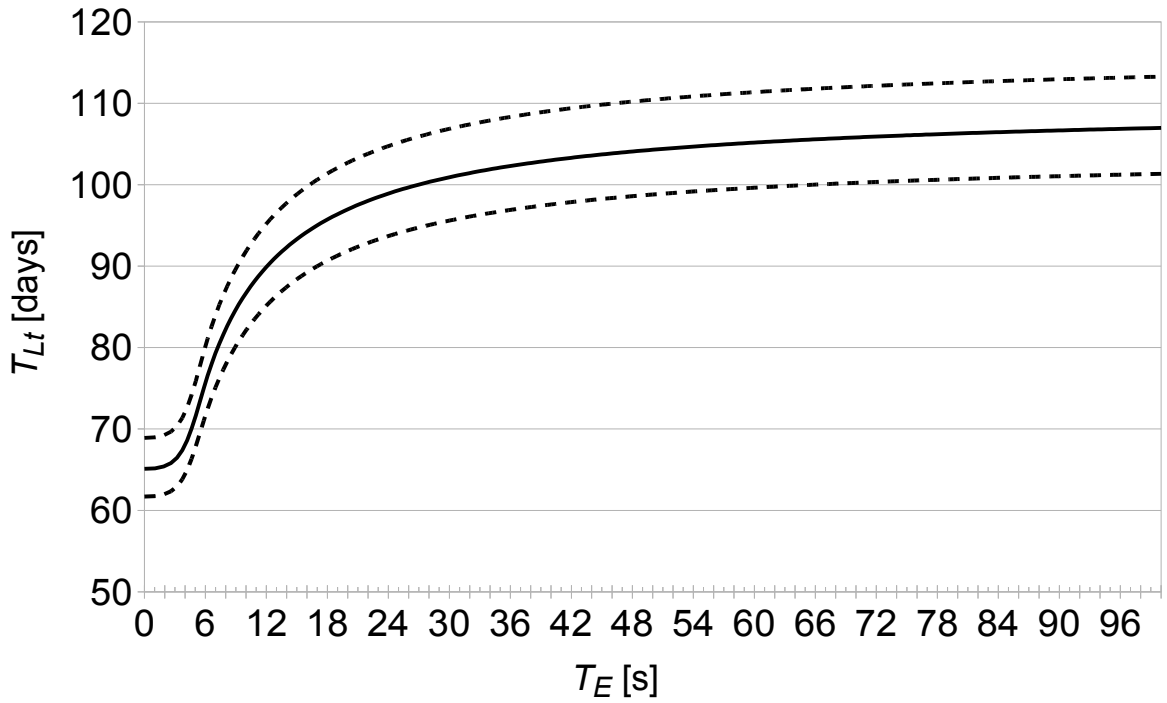


Figure 4.15: Node lifetime vs. average event period. The dashed lines represent the uncertainty interval in which the lifetime may range.

pins undergo to a few transitions from low-to-high voltage and viceversa. This is an indirect advantage brought by the employed vision sensor.

When the system state changes and the decision making process is performed, we expect things not to change dramatically, since this situation simply involves the execution of the algorithm as described in 4.2.1, which is performed in much less time with respect to image transmission. In fact, power consumption raises only by 3% with respect to the previous situation.

We estimated the node lifetime T_{Lt} assuming to power the node with two batteries with a capacity $C = 2200\,\text{mAh}$ at a voltage $V = 3.3\,\text{V}$. Results are shown in Fig. 4.15, where the dashed lines represent the uncertainty interval in which the lifetime may range. This follows from the uncertainty in the measured values of current and voltage drop, according to the law

of uncertainty propagation [77].

The results show that the efficient use of the low-power capabilities of the system's components results in a lifetime of a few months. This is critical for video-surveillance applications, where the system should work for a long time without the need to replace its power source. It should be noted, however, that the lifetime depends on the rate of events to be monitored and therefore is application-specific.

Chapter 5

Conclusion

This thesis has dealt with the development of a camera-based WSN node that consumes very little power. Unlike standard WSNs, in which the stream of data generated by sensing elements such as temperature, pressure and humidity sensors is limited to few bytes per second, WCNs have to cope with a huge amount of data. For example, a low resolution, gray-level camera, running at 10 fps, generates 10 KB of data per second. Each node in the network needs to acquire such data, buffer it in a memory and process it before communicating the result to the other nodes. Moreover, these tasks need to be performed in the very short period of time set by the frame rate. As a consequence, in the past, designers of WCN nodes were forced to equip the nodes with computationally powerful devices in order to meet the timing requirements, at the price of a considerable power consumption. The WCN nodes proposed by researchers consume hundreds of milliwatts, resulting in a network lifetime of few days.

In such a context, we have proposed a method to design an ultra-low-power WCN node. First of all, we have clearly identified each logic element of the node along with its requirements in terms of processing capabilities. These elements are typically the imaging sensor, the control unit, the processing unit and the transceiver. Then, for each logic element, we have

proposed strategies that aim at limiting the impact of the main sources of power consumption to increase the overall lifetime.

The use of standard imaging sensors represents the first source of power consumption, dictating the employment of high performance memories and processors. The key observation here is that standard sensors provide lots of information, most of which is useless for the purposes of the network, and thus the node wastes energy to acquire and buffer data which is then filtered out during the processing phase. To overcome such a problem, we have proposed the employment of CMOS vision sensors, i.e., image sensors that pre-process the data directly on-chip, possibly at the same time as light is integrated. In this way, we generate less data and we move a significant portion of the image processing away from the processing unit. In particular, we carry out those calculations that consist on iteratively applying a certain function throughout the pixels of the image closer to where the information is generated (i.e. in the silicon, in the proximity of the photodiode detecting the incoming light), thus increasing the overall efficiency. Another important feature that the sensor should have is the ability to detect events autonomously, without providing the output image. Thus, the node can operate at high frame rates consuming very little power, since no image is processed outside the chip.

Another source of power consumption is typically represented by the high frequency at which the clock runs within the node. Nevertheless, there are several tasks that we can perform in a relaxed way. Thus, the system energy budget would benefit from mechanisms that allow to dynamically change the clock frequency, according to the complexity of the ongoing tasks, or from architectures in which multiple clock domains are present. In this framework, the control unit plays a major role. By enabling high frequency clock domains only when intensive processing is required, circuits within the node do not switch when not needed, thus achieving considerable

energy saving.

The ability to discriminate situations in which an event of interest is taking place in the scene, with respect to other situations in which there is nothing to worry about, is crucial, too. Our design methodology requires to understand what are the pieces of information contained within the images that are really useful for the application at hand. The node is then trained to identify such situations in a simple and fast way. Thanks to this ability, the system does not run processing intensive algorithms on images that do not represent anything of interest, reducing the power consumed in vain.

The development of power efficient algorithms is another important aspect that must be taken into account. The longer the execution time is, the longer the high frequency clock domains are active. We have proposed to structure the algorithm in two subsequent phases: in the first one, we define the state of the system by extracting a set of features from the images; in the second phase, a decision is carried out on the basis of how the system state has evolved over time. This second phase is performed only when the state changes, therefore we can tune the consumption of power by allowing the system state to assume a smaller or greater set of values.

Finally, with reference to the proposed architecture, we have developed a power model which can be used to determine the maximum power ratings that the node can exhibit in order to achieve a given lifetime. Such a model is based on the expected frequency of the events that we want to detect and on two power configurations. In one case, there is no ongoing event in the scene and the monitoring activity is carried out only by the sensor; in another case, an event is happening and the entire node takes part of the processing activity.

According to the proposed design principles, we have developed a WCN node prototype. Such a node integrates a custom, CMOS vision sensor featuring ultra-low-power consumption, and a flash-based FPGA.

As far as the sensor is concerned, it firstly acquires binary images identifying the edges, i.e., the points of the scene characterized by high contrast. Then, it generates the output by subtracting the acquired image with a previous one stored in a memory internal to the chip itself. This enables to perform background subtraction or motion detection. Two ways of outputting data are provided: the first one is somehow classic, in the sense that the image is read out but compressed in an addressed-based encoding; the second one consists on providing only the number of non-zero pixels present in the output image. As an example, when setting the camera to perform motion detection, the number of non-zero pixels represents a measure of the amount of motion present in the scene. By simply thresholding this value to filter out noise, we get an indicator for the presence or absence of interesting events. In the latter case, we do not need to read out the image, thus avoiding to waste power computing empty images.

The FPGA, built in flash technology, exhibits ultra-low static power consumption. On this PLD, we have implemented a ring oscillator, the control unit, a memory for the image and the processing unit. Two clock domains are present, both derived from the ring oscillator. One runs at a low frequency, for control tasks, and the other runs at a high frequency for processing intensive purposes. The control unit enables the high frequency clock only in certain circumstances, i.e. when the sensor detects a possible event of interest in the scene. In this case, the node operates in the so called Active mode: the sensor reads out the image, the memory is enabled and the processing unit executes the implemented algorithm. On the contrary, when no event are detected in the monitored area, we are in the so called Idle mode: the sensor stops outputting images and keeps monitoring the scene on its own, while the power consuming elements present in the FPGA are disabled.

In the developed prototype, the processing unit represents a people

counter. We have conceived the system to be placed on top of a door, facing downwards, so that people are seen from above while walking through the monitored area. The algorithm is based on two binary features, called Virtual Inductive Loops, each representing a portion of the image that activate when a moving object is detected within it. By concatenating the two features we get the system state, and observing the way in which such state evolves over time we are able to discriminate people entering the scene from those walking in the opposite direction. This is performed by means of probability maps generated in a training phase, which allow to calculate the likelihood with which a given sequence of states represents a person walking by. Given a long sequence of states, the algorithm compares all the possible combinations of events that could have occurred and finds the one that maximizes the overall likelihood. We have represent the problem through a directed multigraph in which we look for the shortest path from a source to a destination. The solution can be found by using a modified version of Dijkstra's dynamic programming algorithm, which we have translated within the FPGA.

Finally, we have performed some experiments in order to prove the validity of our approach. We have measured the power consumed by the node while operating in Idle and Active modes, achieving 2.50 mW and 4.22 mW respectively. In the end, we have reduced power consumption by two orders of magnitude with respect to the other nodes present in the literature. Moreover, by applying the developed power model, and assuming the node powered with a couple of standard batteries with a capacity of 2200 mAh at 3.3 V, we have estimated that its lifetime ranges from a minimum of two months, up to over three months when the frequency of events is very low.

The performed measurements have confirmed that our methodology allows to develop very low-power WCN nodes. We have been able to obtain

these results by reducing the amount of information at the sensor level. Thus, we cannot expect it to provide as many details of the scene as a standard imager would do. In principle, this could prevent the employment of our node for certain applications. Nevertheless, we can cope with this limitation by using a network of heterogeneous nodes, integrating different kinds of sensors and exhibiting different power ratings. In such a framework, ultra-low-power nodes would represent the backbone of the network, since they would be the ones that monitor the scene continuously. When an event of interest occurs, they wake up nodes having enough processing capabilities to perform the desired task. In this way, the most power consuming nodes process the images only when needed, while in the other situations only the ultra-low-power nodes are active. This allows to improve the energy efficiency of the entire network.

Bibliography

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393 – 422, 2002.

[2] S. Soro and W. Heinzelman. A Survey of Visual Sensor Networks. *Advances in Multimedia*, 2009, 2009.

[3] Ali Maleki Tabar, Arezou Keshavarz, and Hamid Aghajan. Smart home care network using sensor fusion and distributed vision-based reasoning. In *VSSN '06: Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks*, pages 145–154, New York, NY, USA, 2006. ACM.

[4] W. Feng, E. Kaiser, W.C. Feng, and M.L. Baillif. Panoptes: scalable low-power video sensor networking technologies. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 1(2):151–167, 2005.

[5] M. Rahimi, R. Baer, O.I. Iroezi, J.C. Garcia, J. Warrior, D. Estrin, and M. Srivastava. Cyclops: in situ image sensing and interpretation in wireless sensor networks. In *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 192–204. ACM, 2005.

[6] R. Kleihorst, A. Abbo, B. Schueler, and A. Danilin. Camera mote with a high-performance parallel processor for real-time frame-based video processing. In *Distributed Smart Cameras, 2007. ICDSC'07. First ACM/IEEE International Conference on*, pages 109–116. IEEE, 2007.

[7] S. Hengstler, D. Prashanth, S. Fong, and H. Aghajan. Mesheye: a hybrid-resolution smart camera mote for applications in distributed intelligent surveillance. In *Proceedings of the 6th international conference on Information processing in sensor networks*, pages 360–369. ACM, 2007.

[8] P. Chen, P. Ahammad, C. Boyer, S.I. Huang, L. Lin, E. Lobaton, M. Meingast, S. Oh, S. Wang, P. Yan, et al. CITRIC: A low-bandwidth wireless camera network platform. In *Distributed Smart Cameras, 2008. ICDSC 2008. Second ACM/IEEE International Conference on*, pages 1–10. IEEE, 2008.

[9] T. Teixeira, D. Lymberopoulos, E. Culurciello, Y. Aloimonos, and A. Savvides. A lightweight camera sensor network operating on symbolic information. In *ACM SenSys Workshop on Distributed Smart Cameras*. Citeseer, 2006.

[10] M. Gottardi, N. Massari, and S.A. Jawed. A 100μw 128×64 pixels contrast-based asynchronous binary vision sensor for sensor networks applications. *Solid-State Circuits, IEEE Journal of*, 44(5):1582 –1592, may 2009.

[11] L. Gasparini, R. Manduchi, M. Gottardi, and D. Petri. An Ultra-Low-Power Wireless Camera Node: Development and Performance Analysis. *Instrumentation and Measurement, IEEE Transactions on,*, 2011. Accepted with minor revision.

[12] Purushottam Kulkarni, Deepak Ganesan, Prashant Shenoy, and Qifeng Lu. Senseye: a multi-tier camera sensor network. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 229–238, New York, NY, USA, 2005. ACM.

[13] OmniVision, Sunnyvale, California, USA. OV6680 SGA product brief.

[14] STMicroelectronics, Geneva, Switzerland. VS6524 VGA Mobile Camera Module data brief.

[15] Microchip Technology, Chandler, Arizona, USA. PIC10F200/202/-204/206 Data Sheet - 6-Pin, 8-bit Flash Microcontrollers.

[16] Freescale Semiconductor, Austin, Texas, USA. MPC5668x Microcontroller Data Sheet.

[17] Marvell Technology Group, Ltd., Santa Clara, California, USA. Marvell PXA3xx Processor Family - Electrical, Mechanical, and Thermal Specification, Data Sheet.

[18] Intel Corporation, Santa Clara, California, USA. Intel Atom Processor E6xx Series Datasheet.

[19] J.S. Lee, Y.W. Su, and C.C. Shen. A comparative study of wireless protocols: Bluetooth, UWB, ZigBee, and Wi-Fi. In *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, pages 46–51. IEEE, 2008.

[20] R. Bruno, M. Conti, and E. Gregori. Mesh networks: commodity multihop ad hoc networks. *Communications Magazine, IEEE*, 43(3):123–131, 2005.

[21] E. Ferro and F. Potorti. Bluetooth and Wi-Fi wireless protocols: a survey and a comparison. *Wireless Communications, IEEE*, 12(1):12–26, 2005.

[22] T. Winkler and B. Rinner. Power aware communication in wireless pervasive smart camera networks. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2009 5th International Conference on*, pages 283–288. IEEE, 2010.

[23] J.A. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile. IEEE 802.15. 4: a developing standard for low-power low-cost wireless personal area networks. *Network, IEEE*, 15(5):12–19, 2002.

[24] ChipCon AS, Oslo, Norway. CC1000 - Single Chip Very Low Power RF Transceiver.

[25] ChipCon AS, Oslo, Norway. CC2420 - 2.4 GHz IEEE 802.15.4/ZigBee-ready RF Transceiver.

[26] ChipCon AS, Oslo, Norway. CC2500 - Low-Cost Low-Power 2.4 GHz RF Transceiver.

[27] ChipCon AS, Oslo, Norway. CC2540 - 2.4-GHz Bluetooth low energy System-on-Chip.

[28] Nordic Semiconductor ASA, Oslo, Norway. nRF8001 - Single-chip Bluetooth low energy solution, Preliminary Product Specification.

[29] Moteiv Corporation, San Francisco, California, USA. Tmote Sky - Ultra low power IEEE 802.15.4 compliant wireless sensor module.

[30] Texas Instruments, Dallas, Texas, USA. MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller.

[31] Memsic Inc., San Jose, California, USA. TelosB Mote Platform.

[32] Memsic Inc., San Jose, California, USA. MICA2 - Wireless Measurement System.

[33] Memsic Inc., San Jose, California, USA. MICAz - Wireless Measurement System.

[34] Atmel Corporation, San Jose, California, USA. 8-bit AVR Microcontroller with 128 KBytes In-System Programmable Flash.

[35] Crossbow Technology, Inc., San Jose, California, USA. High-performance Wireless Sensor Network Node.

[36] L. Nachman, J. Huang, J. Shahabdeen, R. Adler, and R. Kling. Imote2: Serious computation at the edge. In *Wireless Communications and Mobile Computing Conference, 2008. IWCMC'08. International*, pages 1118–1123. IEEE, 2008.

[37] Marvell Technology Group, Ltd., Santa Clara, California, USA. Marvell PXA27x Processor Family - Electrical, Mechanical, and Thermal Specification, Data Sheet.

[38] A. Peleg, S. Wilkie, and U. Weiser. Intel MMX for multimedia PCs. *Communications of the ACM*, 40(1):24–38, 1997.

[39] C.C. Han, R.K. Rengaswamy, R. Shea, E. Kohler, and M. Srivastava. SOS: A dynamic operating system for sensor networks. In *Third International Conference on Mobile Systems, Applications, And Services (Mobisys)*. Citeseer, 2005.

[40] W. Feng, B. Code, E. Kaiser, M. Shea, W. Feng, and L. Bavoil. Panoptes: A scalable architecture for video sensor networking applications. In *Proc. of ACM Multimedia*, pages 151–167. Citeseer, 2003.

[41] Intel Corporation, Santa Clara, California, USA. Intel StrongARM SA-1110 Microprocessor Advance Information Brief Datasheet.

[42] Crossbow Technology, Inc., San Jose, California, USA. Stargate - X-Scale, Processor Platform.

[43] Agilent Technologies, Santa Clara, California, USA. Agilent ADCM-1700 Landscape CIF Resolution CMOS Camera Module - Data Sheet.

[44] Xilinx, Inc., San Jose, California, USA. XC2C256 CoolRunner-II CPLD - Product Specification.

[45] A.A. Abbo, R.P. Kleihorst, V. Choudhary, L. Sevat, P. Wielage, S. Mouy, B. Vermeulen, and M. Heijligers. Xetal-II: a 107 GOPS, 600 mW massively parallel processor for video scene analysis. *Solid-State Circuits, IEEE Journal of*, 43(1):192–201, 2008.

[46] Agilent Technologies, Santa Clara, California, USA. ADNS-3060 High-performance Optical Mouse Sensor Data Sheet.

[47] Agilent Technologies, Santa Clara, California, USA. ADNS-2700 Single Chip USB Optical Mouse Sensor Data Sheet.

[48] Atmel Corporation, San Jose, California, USA. AT91 ARM Thumb-based Microcontrollers.

[49] Z.Y. Cao, Z.Z. Ji, and M.Z. Hu. An image sensor node for wireless sensor networks. In *Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on*, volume 2, pages 740–745. IEEE, 2005.

[50] Samsung Group, Seoul, South Korea. S3C44B0X RISC Microprocessor Product Overview.

[51] I. Downes, L.B. Rad, and H. Aghajan. Development of a mote for wireless image sensor networks. *Proc. of COGnitive systems with Interactive Sensors (COGIS), Paris, France*, 2006.

[52] C. Park and P.H. Chou. eCAM: ultra compact, high data-rate wireless sensor node with a miniature camera. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, pages 359–360. ACM, 2006.

[53] OmniVision, Sunnyvale, California, USA. OV7640 Color CMOS VGA (640 × 480) Camera Chip - Advanced Information, Datasheet.

[54] C. Park and P.H. Chou. Eco: Ultra-wearable and expandable wireless sensor platform. In *Wearable and Implantable Body Sensor Networks, 2006. BSN 2006. International Workshop on*, pages 4–165. IEEE, 2006.

[55] Russ Hersch. 8051 microcontroller faq, September 1997.

[56] Nordic Semiconductor ASA, Oslo, Norway. nRF24E1 - 2.4GHz RF transceiver with embedded 8051 compatible microcontroller and 9 input, 10 bit ADC.

[57] A. Rowe, A. Goode, D. Goel, and I. Nourbakhsh. CMUcam3: an open programmable embedded vision sensor. *International Conferences on Intelligent Robots and Systems*, 2007.

[58] OmniVision, Sunnyvale, California, USA. OV6620 Single-Chip CMOS CIF Color Digital Camera - Advanced Information, Datasheet.

[59] OmniVision, Sunnyvale, California, USA. OV7620 Single-Chip CMOS CIF Color Digital Camera - Advanced Information, Datasheet.

[60] Koninklijke Philips Electronics, Amsterdam, Netherlands. NXP LPC2104/2105/2106 Single-chip 32-bit microcontrollers; 128 kB ISP/-IAP Flash with 64 kB/32 kB/16 kB RAM - Product data.

[61] AverLogic Technologies, Taipei, Taiwan. AL4V8M440 Data Sheet.

[62] R. Mangharam, A. Rowe, and R. Rajkumar. FireFly: a cross-layer platform for real-time embedded wireless networks. *Real-Time Systems*, 37(3):183–231, 2007.

[63] A. Rowe, D. Goel, and R. Rajkumar. Firefly mosaic: A vision-enabled wireless sensor networking system. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 459–468. IEEE, 2007.

[64] A. Kerhet, M. Magno, F. Leonardi, A. Boni, and L. Benini. A low-power wireless video sensor node for distributed object detection. *Journal of Real-Time Image Processing*, 2(4):331–342, 2007.

[65] Atmel Corporation, San Jose, California, USA. 5K - 40K Gates of AT40K FPGA with 8-bit AVR Microcontroller, up to 36K Bytes of SRAM and On-chip JTAG ICE.

[66] OmniVision, Sunnyvale, California, USA. OV9655 1.3 MPixel product brief.

[67] G.J. Pottie and W.J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[68] L. Gasparini, R. Manduchi, M. Gottardi, and D. Petri. Performance analysis of a wireless camera network node. In *Instrumentation and Measurement Technology Conference (I2MTC), 2010 IEEE*, pages 1331–1336. IEEE, 2010.

[69] L. Gasparini, M. Gottardi, D. Macii, and D. Fontanelli. A Low-power Image Acquisition System for Contrast Detection and Processing. In *2011 IMEKO IWADC & IEEE ADC Forum*, 2011. Submitted.

[70] L. Gasparini, M. De Nicola, N. Massari, and M. Gottardi. A micro-power asynchronous contrast-based vision sensor wakes-up on motion.

In *Circuits and Systems, 2008. ISCAS 2008. IEEE International Symposium on*, pages 1040–1043. IEEE, 2008.

[71] B.A. Draper, J.R. Beveridge, A.P.W. Bohm, C. Ross, and M. Chawathe. Accelerated image processing on fpgas. *Image Processing, IEEE Transactions on*, 12(12):1543 – 1551, dec. 2003.

[72] Actel Corporation. Igloo low-power flash fpgas datasheet, 2009.

[73] L. Gasparini, R. Manduchi, and M. Gottardi. An ultra-low-power contrast-based integrated camera node and its application as a people counter. In *Advanced Video and Signal Based Surveillance (AVSS), 2010 Seventh IEEE International Conference on*, pages 547 –554, 29 aug - 1 sep 2010.

[74] E. Viarani. Extraction of traffic information from images at deis. In *Image Analysis and Processing, 1999. Proceedings. International Conference on*, pages 1073 –1076, 1999.

[75] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[76] L. Gasparini, M. Gottardi, N. Massari, D. Petri, and R. Manduchi. FPGA Implementation of a People Counter for an Ultra-Low-Power Wireless Camera Network Node. In *Ph.D. Research in Microelectronics and Electronics (PRIME), 2011 Conference on,*, 2011. Submitted.

[77] IEC BIPM, ISO IFCC, and I. IUPAC. OIML 1995 Guide to the Expression of Uncertainty in Measurement. *International Organization for Standardization, Geneva*, 1995.

[78] T.H. Cormen. *Introduction to algorithms*. The MIT press, 2001.

[79] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. Comparing images using the hausdorff distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(9):850 –863, sep 1993.

[80] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886 –893 vol. 1, 25-25 2005.